The Hieroglyphs
(First Draft) Hieroglyphs: Building Speech Applications Using Sphinx and Related Resources

Arthur Chan       Evandro Gouvêa       Rita Singh
Mosur Ravishankar   Ronald Rosenfield   Yitao Sun
David Huggins-Daines   Mike Seltzer

August 22, 2005
I Before You Start

1 License and use of CMU Sphinx(en), SphinxTrain and CMU-Cambridge LM Toolkit

   1.1 License Agreement of CMU Sphinx II, CMU Sphinx III and SphinxTrain Source Code
       4
   1.2 License Agreement of CMU Sphinx IV
       5
   1.3 License Agreement of the CMU-Cambridge LM Toolkit
       6
   1.4 About this document
       6
   1.5 How to contribute?
       7
   1.6 Version History of Sphinx II
       7
       1.6.1 Sphinx 2.5 Release Notes
       7
       1.6.2 Sphinx 2.4 Release Notes
       8
   1.7 Version History of Sphinx III
       8
       1.7.1 Sphinx 3.5 Release Notes
       8
       1.7.2 Sphinx 3.4 Release Notes
       9
1.8  Version History of Sphinx IV .......................... 10

2  Introduction 13
  2.1  History of Sphinx ................................. 13
    2.1.1  Before Sphinx ............................... 13
    2.1.2  From Sphinx I to Sphinx IV ................. 15
    2.1.3  Diversity of Sphinx Speech Recognition Systems 20
    2.1.4  Summary .................................... 22
  2.2  SphinxTrain ...................................... 22
  2.3  A brief Historical Note of CMU-Cambridge LM Toolkit 23
  2.4  Why do I want to use Sphinx? ..................... 24
  2.5  The Hieroglyphs ................................ 25
  2.6  The Sphinx Developers ............................ 25
  2.7  Which Sphinx(en) should I use? .................. 26
  2.8  Other Open Source activities in Carnegie Mellon 28
    2.8.1  Festival .................................... 29
    2.8.2  CMU Communicator Toolkit .................... 29
    2.8.3  CMUdict, CMUlex ............................ 29
    2.8.4  Speech Databases ............................ 30
    2.8.5  OpenVXML and OpenSALT browser ............. 30
    2.8.6  Miscellaneous ............................... 30
  2.9  How do I get more help? .......................... 30

3  Introduction to Speech Recognition 31
  3.1  The big picture ................................ 31
  3.2  Basic probability ............................... 32
  3.3  What speech recognizers actually do? ............ 34
  3.4  Bayes Rule ..................................... 36
  3.5  Basic of Statistics ............................. 38
  3.6  Problems of Speech Recognition .................. 39
  3.7  Speech Signal in time-domain .................... 41
3.8 Speech Signal as shown in a Spectrogram .......................... 43
3.9 Comparing Things ......................................................... 46
   3.9.1 Comparing Two Time Sequences ............................... 47
   3.9.2 Dynamic Programming ............................................ 49
3.10 Hidden Markov Model .................................................. 51
3.11 Why people used HMM? ................................................ 51
3.12 The three problems of HMM ......................................... 51
3.13 Problem 1: The most likely path problem ......................... 51
3.14 Problem 2: The total likelihood problem .......................... 51
3.15 Problem 3: The parameter estimation problem or training problem .................................................. 51
3.16 Practical acoustic modeling ......................................... 51
3.17 Some basics in language modeling .................................. 52
3.18 Sphinx as a Toolkit of Building Speech Recognition System .... 52

II Tutorial .......................... 53

4 Software Installation .......................... 55
   4.1 Sphinx 3.X ........................................................... 55
      4.1.1 SphinxTrain ..................................................... 56
      4.1.2 CMU LM Toolkit ................................................. 57
      4.1.3 Other tools related to the recognizer ....................... 57
   4.2 FAQ of the installation process .................................. 58
      4.2.1 What if I am a user of Microsoft Windows? ............... 58
      4.2.2 How about other Windows compilers? ....................... 58
      4.2.3 How about other Unix platforms? ........................... 59
      4.2.4 What if I can’t build Sphinx 3.X or SphinxTrain? ........ 59
      4.2.5 What if I am not a fan of C? ................................. 59
      4.2.6 How do I get help for installation? ......................... 59
## 5 Roadmap of building speech recognition system

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Step 1: Design of a Speech Recognition System</td>
<td>62</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Do you really need speech?</td>
<td>62</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Is Your Requirement Feasible?</td>
<td>63</td>
</tr>
<tr>
<td>5.1.3</td>
<td>System Concern</td>
<td>67</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Our recommendation</td>
<td>67</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Summary</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>Step 2: Design the Vocabulary</td>
<td>68</td>
</tr>
<tr>
<td>5.2.1</td>
<td>The System for This Tutorial</td>
<td>68</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Pronunciations of the Words</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Step 3: Design the Grammar for a Speech Recognition System</td>
<td>74</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Coverage of a Grammar</td>
<td>74</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Mismatch between the Dictionary and the Language Model</td>
<td>74</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Interpretation of ARPA N-gram format</td>
<td>75</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Grammar for this tutorial</td>
<td>76</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Conversion of the language model to DMP file format</td>
<td>78</td>
</tr>
<tr>
<td>5.4</td>
<td>Step 4: Obtain an Open Source Acoustic Model</td>
<td>79</td>
</tr>
<tr>
<td>5.4.1</td>
<td>How to get a model?</td>
<td>79</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Are there other models?</td>
<td>81</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Why Acoustic Models are so Rare?</td>
<td>81</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Interpretation of Model Formats</td>
<td>81</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Summary</td>
<td>86</td>
</tr>
<tr>
<td>5.5</td>
<td>Step 5: Develop a speech recognition system</td>
<td>87</td>
</tr>
<tr>
<td>5.5.1</td>
<td>About the Sphinx 3’s Package</td>
<td>87</td>
</tr>
<tr>
<td>5.5.2</td>
<td>A batch mode system using decode and decode_anytopo: What could go wrong?</td>
<td>90</td>
</tr>
<tr>
<td>5.5.3</td>
<td>A live mode simulator using live_pretend</td>
<td>95</td>
</tr>
<tr>
<td>5.5.4</td>
<td>A Live-mode System</td>
<td>95</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Could it be Faster?</td>
<td>97</td>
</tr>
<tr>
<td>5.6</td>
<td>Step 6: Evaluate your own system</td>
<td>99</td>
</tr>
<tr>
<td>5.6.1</td>
<td>On-line Evaluation</td>
<td>99</td>
</tr>
</tbody>
</table>
5.6.2 Off-line Evaluation ........................................... 100
5.7 Interlude ......................................................... 101
5.8 Step 7: Collection and Processing of Data ...................... 101
5.8.1 Determine the amount of data needed .......................... 102
5.8.2 Record the data in appropriate condition: environmental noise .................................................. 102
5.9 Step 8: Transcription of data ..................................... 104
5.10 Step (7 & 8)b: If you don't want to sweat ....................... 105
5.11 Step (7 & 8)c: If you don't want to sweat and you have no money ...................................................... 106
5.12 Step 9: Training of acoustic model. Follow a recipe .......... 107
5.13 Step 10: Training of acoustic model. Do it from scratch .... 108
5.14 Step 11: Trace the source code of Sphinx .................... 109
5.15 How do I get more help? ........................................ 110

III CMU Sphinx Speech Recognition System ....................... 111

6 Sphinx’s Front End ................................................. 113
6.1 Introduction ...................................................... 113
6.2 Block Diagram .................................................. 114
6.3 Front End Processing Parameters ................................ 115
6.4 Detail of Front End processing ................................... 115
6.4.1 Pre-emphasis .................................................. 115
6.4.2 Framing ....................................................... 116
6.4.3 Windowing .................................................... 116
6.4.4 Power Spectrum .............................................. 116
6.4.5 Mel Spectrum ................................................ 116
6.4.6 Mel Cepstrum ................................................ 116
6.5 Mel Filter Banks .................................................. 117
6.6 The Default Front End Parameters .............................. 117
6.7 Computation of Dynamic Coefficients ......................... 118
6.7.1 Generic and simplified notations in SphinxTrain . . . . 119
6.7.2 Formulae for Dynamic Coefficient . . . . . . . . . . . . . 119
6.7.3 Handling of Initial and Final Position of an Utterance . 119
6.8 Cautions in Using the Front End . . . . . . . . . . . . . . . . 120
6.9 Compatibility of Front Ends in Different Sphinxen . . . . . . 120
6.10 Cepview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120

7 General Software Description of SphinxTrain 123
7.1 On-line Help of SphinxTrain . . . . . . . . . . . . . . . . . . . . 123
7.2 Basic Acoustic Models Format in SphinxTrain . . . . . . . . 125
7.3 Sphinx data and model formats . . . . . . . . . . . . . . . . . . 126
  7.3.1 Sphinx II data formats . . . . . . . . . . . . . . . . . . . . 126
  7.3.2 Sphinx II Model formats . . . . . . . . . . . . . . . . . . . 126
  7.3.3 Sphinx III model formats . . . . . . . . . . . . . . . . . . . 132
  7.3.4 Sphinx IV model formats . . . . . . . . . . . . . . . . . . . 135
7.4 Software Architecture of SphinxTrain . . . . . . . . . . . . . . 136
  7.4.1 General Description of the C-based Applications . . . . . 136
  7.4.2 General Description of the Perl-based Tool . . . . . . . . . 137

8 Acoustic Model Training 139
8.1 Basic Concepts of Training . . . . . . . . . . . . . . . . . . . . . 140
  8.1.1 Flat Initialization . . . . . . . . . . . . . . . . . . . . . . . . 140
  8.1.2 Continous and Semi-continuous HMM . . . . . . . . . . . 140
  8.1.3 Baum-Welch algorithm . . . . . . . . . . . . . . . . . . . . . 140
  8.1.4 Decision Tree tying . . . . . . . . . . . . . . . . . . . . . . . 140
  8.1.5 Mixture Growing . . . . . . . . . . . . . . . . . . . . . . . . 140
8.2 Before you start . . . . . . . . . . . . . . . . . . . . . . . . . . . 140
  8.2.1 The general-procedure chart . . . . . . . . . . . . . . . . . 140
  8.2.2 Modeling context-dependent phones with untied states: some memory requirements . . . . 141
  8.2.3 Data preparation . . . . . . . . . . . . . . . . . . . . . . . . . 142
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.4 The set of base and higher order feature vectors</td>
</tr>
<tr>
<td>8.2.5 Force-alignment</td>
</tr>
<tr>
<td>8.3 Training Fully Continuous Hidden Markov Model</td>
</tr>
<tr>
<td>8.3.1 Creating the CI model definition file</td>
</tr>
<tr>
<td>8.3.2 Creating the HMM topology file</td>
</tr>
<tr>
<td>8.3.3 Flat initialization of CI model parameters</td>
</tr>
<tr>
<td>8.3.4 Training CI models</td>
</tr>
<tr>
<td>8.3.5 Creating the CD untied model definition file</td>
</tr>
<tr>
<td>8.3.6 Flat initialization of CD untied model parameters</td>
</tr>
<tr>
<td>8.3.7 Training CD untied models</td>
</tr>
<tr>
<td>8.3.8 Building decision trees for parameter sharing</td>
</tr>
<tr>
<td>8.3.9 Pruning the decision trees</td>
</tr>
<tr>
<td>8.3.10 Creating the CD tied model definition file</td>
</tr>
<tr>
<td>8.3.11 Initializing and training cd tied gaussian mixture models</td>
</tr>
<tr>
<td>8.4 Training semi-continuous models</td>
</tr>
<tr>
<td>8.4.1 Difference between training continuous HMM and semi-continuous HMM</td>
</tr>
<tr>
<td>8.4.2 Vector quantization</td>
</tr>
<tr>
<td>8.4.3 Deleted interpolation</td>
</tr>
<tr>
<td>8.5 Using SCHMM’s decision trees to train a FCHMM</td>
</tr>
<tr>
<td>8.6 Updating or adapting existing models sets</td>
</tr>
<tr>
<td>8.7 Training Multilingual Models</td>
</tr>
<tr>
<td>8.8 The training lexicon</td>
</tr>
<tr>
<td>8.9 10 Common Pitfalls of Using SphinxTrain</td>
</tr>
</tbody>
</table>

9 Language Model Training 187

<table>
<thead>
<tr>
<th>Language Model Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Installation of the Toolkit</td>
</tr>
<tr>
<td>9.2 Terminology and File Formats</td>
</tr>
<tr>
<td>9.3 Typical Usage</td>
</tr>
<tr>
<td>9.4 Discounting Strategies</td>
</tr>
<tr>
<td>9.4.1 Good Turing discounting</td>
</tr>
</tbody>
</table>

This page was generated with the help of DOC++
http://docpp.sourceforge.net
9.4.2 Witten Bell discounting .......................... 192
9.4.3 Linear discounting .............................. 192

10 Search structure and Speed-up of the speech recognizer 193
10.1 Introduction ........................................ 193
10.2 Sphinx 3X's recognizer general architecture .......... 194
10.3 Importance of tuning ................................ 195
10.4 Sphinx 3.X's decode_anytopo (or s3slow) .............. 195
   10.4.1 GMM Computation ............................. 196
   10.4.2 Search structure ............................... 196
   10.4.3 Treatment of language model .................... 197
   10.4.4 Triphone representation ......................... 197
   10.4.5 Viteri Pruning ................................ 197
   10.4.6 Search Tuning .................................. 197
   10.4.7 2-nd pass Search ............................... 198
   10.4.8 Debugging ..................................... 199
10.5 Sphinx 3.X's decode (aka s3 fast) ...................... 199
10.6 Architecture of Search in decode ...................... 199
   10.6.1 Initialization ................................ 200
   10.6.2 Lexicon representation ......................... 201
   10.6.3 Language model ................................ 203
   10.6.4 Pruning ....................................... 206
   10.6.5 Phoneme look-ahead ............................ 209
10.7 Architecture of GMM Computation in decode .......... 209
   10.7.1 Frame-level optimization ....................... 209
   10.7.2 GMM-level optimization ......................... 210
   10.7.3 Gaussian-level optimization : VQ-based Gaussian Selection and SVQ-based Gaussian Selection ............. 210
   10.7.4 Gaussian/Component-level optimization : Sub-vector quantization ............................... 211
   10.7.5 Interaction between different level of optimization ............ 212
   10.7.6 Related tools, gs_select, gs_view, gausubvq ........ 212
12.3 Using the live mode recognizer ........................................ 232
  12.3.1 Programs .......................................................... 232
12.4 Common problems in setting up the live-mode speech recognizer ........................................ 235
12.5 livedecodeAPI.c doc generated by doc++ .............................. 236

13 Developing Sphinx ................................................................ 249
  13.1 Reasons of Developing an Open Source Speech Recognizer .... 249
  13.2 Internal of Sphinx3 ....................................................... 251
  13.3 Internal of SphinxTrain .................................................. 251
  13.4 Challenges of development of Sphinx and open source speech recognition in future ....................... 251
    13.4.1 The issue of Model Training .................................... 251
    13.4.2 The issue of Code duplication ................................. 253
    13.4.3 The separation of Sphinx 3 and SphinxTrain .............. 253
    13.4.4 The issue of user interface complexity ..................... 253
    13.4.5 The issue of lack of researchers/developers’ Education 253
    13.4.6 Is English the only language? ............................... 253
    13.4.7 Is PC the only platform? ....................................... 253
    13.4.8 Is Sphinx the only recognizer ............................... 253
  13.5 How to contribute? ...................................................... 253
    13.5.1 A patch .......................................................... 253
    13.5.2 A customize version .......................................... 253
    13.5.3 A merge ....................................................... 253
  13.6 For future maintainer .................................................... 253

A Command Line Information .................................................. 255
  A.1 Sphinx III Decoders .................................................... 255
    A.1.1 decode .......................................................... 255
    A.1.2 livedecode ...................................................... 263
    A.1.3 livepretend ...................................................... 272
    A.1.4 gausubvq ....................................................... 281
| A.1.5 | align | 284 |
| A.1.6 | allphone | 284 |
| A.1.7 | dag | 284 |
| A.1.8 | astar | 284 |
| A.1.9 | cepview | 284 |
| A.2 | SphinxTrain: Executables | 285 |
| A.2.1 | agg_seg | 285 |
| A.2.2 | bldtree | 289 |
| A.2.3 | bw | 292 |
| A.2.4 | cp_parm | 297 |
| A.2.5 | delint | 299 |
| A.2.6 | dict2tri | 301 |
| A.2.7 | inc_comp | 303 |
| A.2.8 | init_gau | 305 |
| A.2.9 | init_mixw | 308 |
| A.2.10 | kmeans_init | 310 |
| A.2.11 | map_adapt | 314 |
| A.2.12 | makequests | 316 |
| A.2.13 | mixw_interp | 318 |
| A.2.14 | mk_flat | 320 |
| A.2.15 | mk_mdef_gen | 322 |
| A.2.16 | mk_mllr_class | 324 |
| A.2.17 | mk_model_def | 326 |
| A.2.18 | mk_s2cb | 328 |
| A.2.19 | mk_s2hmm | 330 |
| A.2.20 | mk_s2phone | 332 |
| A.2.21 | mk_s2phonemap | 334 |
| A.2.22 | mk_s2sendump | 336 |
| A.2.23 | mk_s3gau | 338 |
| A.2.24 | mk_s3mix | 340 |
| A.2.25 | mk_s3tmat | 342 |
## Contents

A.2.26 mk_ts2cb ........................................ 344
A.2.27 mllr_solve ...................................... 346
A.2.28 mllr_transform .................................. 348
A.2.29 norm ............................................ 350
A.2.30 param_cnt ....................................... 352
A.2.31 printp ........................................... 354
A.2.32 prunetree ........................................ 356
A.2.33 tiestate .......................................... 358
A.2.34 wave2feat ....................................... 361
A.2.35 QUICK_COUNT ................................... 366

A.3 SphinxTrain: Perl Training Scripts .................. 368
   A.3.1 00.verify/verify_all.pl ...................... 368
   A.3.2 01.vector_quantize/slave.VQ.pl .......... 368
   A.3.3 02.ci_schmm/slave_convg.pl .............. 368
   A.3.4 03.makeuntiedmdef/make_untied_mdef.pl .... 368
   A.3.5 04.cd_schmm_untied/slave_convg.pl ...... 368
   A.3.6 05.buildtrees/slave.treebuilder.pl ...... 368
   A.3.7 06.prunetree/prunetree.pl ............... 368
   A.3.8 07.cd_schmm/slave_convg.pl ............ 368
   A.3.9 08.deleted-interpolation/deleted_interpolation.pl 368
   A.3.10 Summary of Configuration Parameter in script level.. 368
   A.3.11 Notes on parallelization .................... 368

A.4 CMU LM Toolkit .................................... 369
   A.4.1 binlm2arpa .................................... 369
   A.4.2 evallm ......................................... 370
   A.4.3 idngram2lm .................................... 372
   A.4.4 idngram2stats ................................ 374
   A.4.5 interpolate .................................... 375
   A.4.6 mergeidngram .................................. 377
   A.4.7 ngram2mgram .................................. 377
   A.4.8 text2idngram .................................. 378
B Frequently Asked Questions 383
B.1 Introduction 383
B.2 Data-preparation for training acoustic models. 383
B.3 Telling size of speech corpus in hours 383
B.4 Force alignment 384
B.5 Selecting modeling parameters. 389
B.6 Rule of thumb of choosing parameters 389
B.7 Feature computation 391
B.8 Modeling filled pauses and non-speech events. 393
B.9 Training speed. 396
B.10 Questions specific to log files. 397
B.11 Vector-quantization for discrete and semi-continuous models 400
B.12 Updating existing models 403
B.13 Utterance, word and phone segmentations 404
B.14 Force-alignment (Viterbi alignment) 405
B.15 Baum-Welch iterations and associated likelihoods 408
B.16 Dictionaries, pronunciations and phone-sets 411
B.17 Decision tree building and parameter sharing 414
B.18 Post-training disasters 415
B.19 Why is my recognition accuracy poor? 417
B.20 Interpreting SPHINX-II file formats. 419
B.21 Interpreting SPHINX-III file formats. 422
B.22 Hypothesis combination 424
B.23 Language model 426
B.24 Training context-dependent models with untied states 427
B.25 Acoustic likelihoods and scores 428
B.26 Decoding problems ........................................ 431
B.27 Why Sphinx III’s performance is poorer than recognizer X? . . 432
List of Figures

3.1 Waveforms for Hello1 .................................................. 41
3.2 Waveforms for Hello2 .................................................. 42
3.3 Spectrogram for Hello1 .................................................. 44
3.4 Spectrogram for Hello2 .................................................. 45
3.5 Search Graph for matching “a b c d” and “a b c”. The dark line represents the final alignment ........................................ 50

5.1 Resource used by the decoders ................................. 91
List of Tables

A.1 Typical parameter values for common sampling rates . . . . . 361
CMU Sphinx (or Sphinx) is one of the most powerful open source speech recognition systems in the world. In recent years, the Sphinx developers start to extend the scope of the CMU Sphinx from maintainence of tools to supporting various resource that enable developers to build successful speech applications.

In the past, “CMU sphinx” used to mean just the speech recognizer. Or the process of tranforming acoustic waveforms into word. (or Speech-to-Text). In these days, it is recognized that the process of building a speech application involved not only the recognizer but also several interrelated resources. These includes the acoustic model trainer, the acoustic and language models. It is also recognized that building an application with speech input requires special technique and knowledge.

As CMU Sphinx and its resource grows in last 5 years, it is an alarming fact that the most challenging problem that a lot of developers is facing is still lack of documentation. Or the documentation was scattered into multiply web sites.

This makes the use of Sphinx could only restricted to users with long year of experience in programming or speech recognition research. The developers of Sphinx want to change this situation by introducing a single book that record most, if not all, knowledge of using Sphinx and its related resource to build speech applications. Following the naming convention of CMU to use mythological object as project name. This effort was code named “Hieroglyph”.

The name “Hieroglyphs” symbolizes an important aspect of learning to use a speech recognition toolkit. I.e. it requires extensive knowledge. It is not an exaggeration that expert of any speech recognition system is usually expert in several seemingly unrelated fields, phonetics, linguistics, signal processing, pattern recognition, computer algorithms and artificial intelligence. The knowledge required might be as deep as the knowledge required to learn to understand a type of Hieroglyphs.
We hope that “Hieroglyph” will also later turn into an encyclopedia-type of text. That allows both beginners and experts to learn the essential facts of using Sphinx and its related tools.

### The writing process and current status

There are vast amount of knowledge of using Sphinx, the fact that multiple versions of Sphinx(en) exist also steepen the users learning curve. Therefore, we chose to use one single recognizer as a starting point of the material. The goal of the first edition is therefore to create a single document that a developer could use

**Sphinx 3 + SphinxTrain + CMU-Cambridge LM Toolkit**

to create an application. We chose to use Sphinx 3 mainly because it is the active branch which CMU is maintaining at the time. We also found that the combination is more coherent because all three tools are written in same programming language.

Even with the more limited scope, the editor still found that there are 15 chapters of material (2 of them are appendices) are required to be completed. It is fortunate that the original of all previous documents of each part of Sphinx are willing to give permission to either reprint or re-use their material. It makes the writing process much easier.

As at Jun, 2005, the first drafts of 9 chapters have already fully completed.

As at the end of Aug, 2005, the first draft of Chapter 7, 8, 9 are also completed.

Here are descriptions of the completed chapters, material they references and their current status.

In Chapter 1, the license and version history of CMU sphinx(en), SphinxTrain and CMU-Cambridge LM Toolkit will be described. This chapter is completed.

Chapter 2 describes the history of development of CMU Sphinx, SphinxTrain and CMU-Cambridge LM Toolkit. The difference between different versions of Sphinx and a brief guideline on how to choose which recognizer to use. The part of history of Sphinx largely referenced Dr Rita Singh’s “History of Sphinx” document. This chapter is completed.

Chapter 4 describes how to install software of Sphinx. This part is completed.
Chapter 6 provides information of how Sphinx carries out feature extraction and dynamic coefficient computation. This part is largely referenced from Dr. Mike Seltzer's document of “Sphinx III front end Specification”. This content of this part is almost completed but required proof-reading. We also feel that it is necessary to add details such as CMN, variance normalization and AGC into this chapter.

Chapter 7 provides a general description of the software package of our training process. It is almost completed. However, some details such as description of Sphinx III file format and how the headers look like are still missing.

Chapter 8 provides a detail look on how acoustic model could be training for both continuous HMM and semi-continuous HMM. It is largely adapted from Dr. Rita Singh’s web page for “instruction of training”. This part is largely completed. However, it still lacks of example that could help users to solve a problem when they have a real-life situation. In the next revision, we will focus on solving these problems.

Chapter 9 provides a manual for using CMU-Cambridge language model toolkit. The material is largely adapted from Dr. Philip Clarkson’s web page on the instruction.

Chapter 10 provides detail description on how the search of Sphinx III works. This part reference Dr. Mosur Ravishankar’s tutorial on “Sphinx III” and my own presentation on “From Sphinx 3.3 to Sphinx 3.4” and “From Sphinx 3.4 to Sphinx 3.5”. This part is largely completed. However, important detail description on how lexical tree, n-gram search and context-dependent triphones work on decode and decode_anytopo is still missing at the current stage.

Chapter 11 discusses how to use the speaker adaptation facilities began to be provided by SphinxTrain and sphinx III from 2004. This part is mostly completed. However, we might still want to add detail on how better alignment on the transcription could provide larger gain in speaker adaptation.

Chapter 12 discusses how to make use of live-mode APIs of Sphinx III to develop speech application. The backbone of this chapter is completed. It was largely reference Yitao Sun’s code documentation that was automatically generated by doc++. However, in 2005, the Sphinx developer decide to switch from doc++ to doxygen. Therefore, we will expect to replace the current documentation very soon.

Appendix A provides all command line information for all tools in Sphinx III, SphinxTrain and CMU LM Toolkit. This part relies on automatic generation of the tex file from the SphinxTrain and Sphinx III’s command line. The SphinxTrain tool part is completed but need to be updated. The
SphinxTrain script, Sphinx III and CMU LM Toolkit are not yet completed.

Appendix B lists all frequently asked questions that were collected by Dr. Rita Singh. This part largely copied from the page from cmusphinx.org which is first collected by Dr. Rita Singh and is no maintained by Dr. Evan-dro Gouvêa. This part is completed. However, updates will be necessary when new questions come up in the developers mind.

**Current focus of writing**

The editor is currently focused on working chapter 5 which provides detail description on how to use the Sphinx System from scratch. After completion of it, the document will be released in the editor’s web page again.

**Furture Plan**

The coexistence of Sphinx II, Sphinx III and Sphinx IV and their extensive usage in industry are perhaps a sign of the creativity of workers of speech recognition. We hope that this trend could be continued. Each Sphinx represent a spectrum of speech application that developers or researchers want to build. Therefore, it is fair to explain their usage in details in this document as well.

As we mentioned, the writing of edition one of Hieroglyph will only cover Sphinx 3 which happens to be the active branch. After its completion, we will extend it to cover Sphinx II and Sphinx IV’s usage as well.

**For the users**

This manual was created to aide the users in dealing with difficulties of using Sphinx and its related tools. To the user, we would hope to you can accept our appology. It is probably reasonable for us to include so much material in the first place. Serious knowledge of building an HMM-based speech recognizer could hardly be written using only 10 pages. Making a serious speech recognition is a very difficult task. Even with advanced tools like Sphinx, it could still require high-level of skill and perhaps more importantly tremendous amount of knowledge as well as patience.
For user who has no idea of what a speech recognizer is. Chapter 3 provides a good start for speech recognition.

We recommend the users without basic knowledge of HMM to at least go through the background material in Chapter ?? From our experience in answering user's mail in Sphinx's Forum, it is close to impossible to use a HMM-based speech recognizer without knowing the basic of what is HMM. Even the user could use the recognizer to run decoding for once. They would soon get lost in further development because they have no idea on what Viterbi algorithm is. We also suggest them to read Chapter 3

For users who have basic knowledge of HMM, we will recommend them to browse through Chapter 2 and then follow the instruction as given in Chapter 5. Chapter 2 has a nice interview what resource the CMU Sphinx Group to build a speech recognition system. Whereas Chapter 5 can serve as a reference to build the first system.

For users who already have knowledge in Sphinx, we will recommend you to keep track of the new release of this document and the release and backward compatibility notes we augment in every minor version updates.

We are also well aware that the document itself could have mistakes and we are always welcome for users' feedback on how to improve it. Just send your opinions to the current maintainers, the maintainers: Arthur Chan, Evandro Gouvêa and Alan Black.

Arthur Chan
The Editor (who only cuts and pastes)
Written at May 2005
Part I

Before You Start
Chapter 1

License and use of CMU Sphinx(en), SphinxTrain and CMU-Cambridge LM Toolkit

Author: Arthur Chan, Editor: Arthur Chan

- CMU Sphinx II, CMU Sphinx III and SphinxTrain have been developed by researchers and developers at Carnegie Mell University (CMU).

- CMU Sphinx IV have been co-developed by research and developers at Carnegie Mell University (CMU), Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

- CMU-Cambridge LM Toolkit was co-developed by researchers at CMU and Cambridge University.

If you are interested in Sphinx, your right regarding the code and binaries in Sphinx, SphinxTrain and CMU-Cambridge LM Toolkit must be your major concern. Here is something you may want to know:

1. **Copying of Sphinx II, Sphinx III, Sphinx IV, SphinxTrain** The code and binary of CMU Sphinx(en) are *free* for commercial/non-commercial users with or without modification. For more details, please check the license terms below.
2. **Copying of CMU-Cambridge LM Toolkit** The code of the CMU-Cambridge LM toolkit is made available for research purposes only. It may be redistributed freely for this purpose.

3. **Academic license** We are not using GPL-ed license. Hence, you are not obligated to distribute your source code.

4. **Warranty**

Although the license of CMU Sphinx (II, III and IV) and SphinxTrain’s source code are free and the CMU-Cambridge LM toolkit’s source code is free for research purpose, we do not provide any warranty and support at all. However, the maintainers of Sphinx are always willing to discuss and help developers to build applications using Sphinx. As of August 22, 2005, the maintainer is Arthur Chan, you are welcome to talk to him.

Sections 1.1 and 1.3 contain the full license terms for the software.

### 1.1 License Agreement of CMU Sphinx II, CMU Sphinx III and SphinxTrain Source Code

Copyright (c) 1995-2004 Carnegie Mellon University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This work was supported in part by funding from the Defense Advanced Research Projects Agency and the National Science Foundation of the United States of America, and the CMU Sphinx Speech Consortium.

THIS SOFTWARE IS PROVIDED BY CARNEGIE MELLON UNIVERSITY “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY NOR ITS EMPLOYEES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2 License Agreement of CMU Sphinx IV


Portions Copyright 2002-2004 Sun Microsystems, Inc.

Portions Copyright 2002-2004 Mitsubishi Electric Research Laboratories.

All Rights Reserved. Use is subject to license terms.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Original authors' names are not deleted.

4. The authors' names are not used to endorse or promote products derived from this software without specific prior written permission.

This work was supported in part by funding from the Defense Advanced Research Projects Agency and the National Science Foundation of the United States of America, the CMU Sphinx Speech Consortium, and Sun Microsystems, Inc.

CARNEGIE MELLON UNIVERSITY, SUN MICROSYSTEMS, INC., MITSUBISHI ELECTRONIC RESEARCH LABORATORIES AND THE CONTRIBUTORS TO THIS WORK DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY, SUN MICROSYSTEMS, INC., MITSUBISHI ELECTRONIC RESEARCH LABORATORIES NOR THE CONTRIBUTORS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

1.3 License Agreement of the CMU-Cambridge LM Toolkit

Copyright (C) 1996, Carnegie Mellon University, Cambridge University, Ronald Rosenfeld and Philip Clarkson.

All rights reserved.

This software is made available for research purposes only. It may be redistributed freely for this purpose, in full or in part, provided that this entire copyright notice is included on any copies of this software and applications and derivations thereof.

This software is provided on an "as is" basis, without warranty of any kind, either expressed or implied, as to any matter including, but not limited to warranty of fitness of purpose, or merchantability, or results obtained from use of this software.

1.4 About this document

We hope that this document itself can be distributed so that its quality can be improved. Hence, the following permissions are granted to the users:

- Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

- Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

- Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified
1.5 How to contribute?

Sphinx is written for all researchers and developers of the world. We chose to use the X11 style license because this will allow users to have higher flexibility in incorporating the code.

To truly achieve this goal, your participation is necessary. Due the limitation of resources, Sphinx’s performance still has a certain gap with state-of-the-art system. Your participation is necessary to refine and perfect this software. We would also like to collect models for different languages and in different conditions. This will allow users/developers in the world to develop speech recognition systems easily.

The software is still under heavy development to know which part of the software requires change. You are visit the CMU Sphinx web page:

http://cmusphinx.org

to learn about the open projects within the Sphinx group.

Please contact the maintainers: Arthur Chan, Evandro Gouvêa and Alan Black if you are interested in contributing to the Sphinx speech recognition system.

1.6 Version History of Sphinx II

1.6.1 Sphinx 2.5 Release Notes

This release (sphinx2-0.5) includes support for language model defined as a Finite State Grammar (FSG), as well as bug fixes.

The FSG implementation supports:

- an application can chose at run time which FSG to use, from a list provided by the user (also supported with the statistical LM).

- the FSG can be passed to the decoder as a data structure instead of as a file name.
• FSG automatically inserts multiple pronunciations and noise models, if provided.

Among the fixes, the executables and libraries created with MS Visual Studio are placed in a more sensible location.

It should work in several flavors of Unix (including Linux and Solaris), and Windows NT or later.

1.6.2 Sphinx 2.4 Release Notes

CMU Sphinx is a speech recognition system.

This version includes a FreeBSD-style license (2 conditions), and numerous fixes. It should work for Linux, Solaris, various Unixes, and Windows under both Visual Studio and cygwin.

1.7 Version History of Sphinx III

1.7.1 Sphinx 3.5 Release Notes

New features of this release:

• Live-mode APIs are stable and are now officially released.

• Windows version of live-mode decoder based on the new APIs is released.

• Live-mode decoder simulator (livepretend) based on the new APIs is still under development. We expect it to be stable in Sphinx 3.6.

• Mean transformation-based adaptation is now supported.

• The feature extraction library of sphinx3 is now EXACTLY the same as SphinxTrain

• Four of the s3.0 (flat decoder) tools are now incorporated in Sphinx 3.x.
  – align - a time aligner
  – astar - an N-best genertor
- allphone - a phoneme recognizer
- dag - a best-path finder for a lattice

SphinxTrain has been tagged to match the release 0.5 of sphinx3. One can retrieve the matching SphinxTrain by the command:

$ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/cmusphinx/ co -r SPHINX3.5_CMUINTERNAL_RELEASE SphinxTrain

New features of this release:

- The feature extraction libraries of SphinxTrain is now EXACTLY the same as sphinx3.
- New tools introduced
  - mllr_solve, given the EM posterior probabilities of the mean and regression class definition, this routine estimates regression matrices for each class.
  - mllr_transform, given a transformation matrix, this tool applies the transformation to the mean vector of the models.
- The command line interface for all SphinxTrain tools are unified.

1.7.2 Sphinx 3.4 Release Notes

1. Sphinx 3.4 has a re-written the Gaussian computation routine. The following techniques are implemented.
   - Naive and model-based down-sampling such that some of the frames are computed.
   - Context-independent phone-based Gaussian mixture model Selection.
   - Two Gaussian selection algorithms including Vector Quantizer-based Gaussian selection and Sub-vector quantizer-based Gaussian selection.
   - Sub-vector quantization.

2. Phoneme lookahead based on three different types of heuristics are implemented.

3. Sphinx 3.3 front-end problem is fixed. Current front-end supports the original s3.1x39 feature and also the standard 1s_c_d_dd features.
4. Live mode recognizer’s core dump problem is fixed.

5. sphinx-test and sphinx-simple should work as it is.

6. Visual C++ .dsw file is moved to upper most level of the code, a project is also built for the program decode in windows.

7. README and several compilation document is now included in the distribution.

1.8 Version History of Sphinx IV

Sphinx 4 1.0 beta

In this release, we have provided the following new features and improvements over the 0.1 alpha release:

- Confidence scoring
- Dynamic grammar support
- JSGF limitations removed
- Improved performance for large, perplex JSGF grammars
- Filler support for JSGF Grammars
- Out-of-grammar utterance rejection
- Narrow bandwidth acoustic model
- WSJ5K Language model
- More demonstration programs
- Better control over microphone selection
- Lots of bug fixes

Sphinx-4 is a state-of-the-art, speaker-independent, continuous speech recognition system written entirely in the Java programming language. It was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from
the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

The design of Sphinx-4 is based on patterns that have emerged from the design of past systems as well as new requirements based on areas that researchers currently want to explore. To exercise this framework, and to provide researchers with a "research-ready" system, Sphinx-4 also includes several implementations of both simple and state-of-the-art techniques. The framework and the implementations are all freely available via open source under a very generous BSD-style license.

With the 1.0 beta release, you get the complete Sphinx-4 source tree along with several acoustic and language models capable of handling a variety of tasks ranging from simple digit recognition to large vocabulary n-Gram recognition.

Because it is written entirely in the Java programming language, Sphinx-4 can run on a variety of platforms without requiring any special compilation or changes. We’ve tested Sphinx-4 on the following platforms with success: the Solaris 9 Operating System on the SPARC platform, Mac OS X 10.3.5, RedHat 9.0, Fedora Core 1, Microsoft Windows XP, and Microsoft Windows 2000.

Please give Sphinx-4 1.0 beta a try and post your questions, comments, and feedback to one of the CMU Sphinx Forums:

https://sourceforge.net/forum/?group_id=1904

We can also be reached at cmusphinx-contacts@lists.sourceforge.net.

Sphinx 4-alpha

Sphinx-4 is a state-of-the-art, speaker-independent, continuous speech recognition system written entirely in the Java programming language. It was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and the Massachusetts Institute of Technology (MIT).

The design of Sphinx-4 is based on patterns that have emerged from the design of past systems as well as new requirements based on areas that researchers currently want to explore. To exercise this framework, and to provide researchers with a "research-ready" system, Sphinx-4 also includes several implementations of both simple and state-of-the-art techniques. The framework and the implementations are all freely available via open source under a very generous BSD-style license.
With the Alpha 0.1 release, you get the complete Sphinx-4 source tree along with several acoustic and language models capable of handling a variety of tasks ranging from simple digit recognition to large vocabulary n-Gram recognition.

Because it is written entirely in the Java programming language, Sphinx-4 can run on a variety of platforms without requiring any special compilation or changes. We’ve tested Sphinx-4 on the following platforms with success: the Solaris 9 Operating System on the SPARC platform, Mac OS X 10.3.3, RedHat 9.0, Fedora Core 1, Microsoft Windows XP, and Microsoft Windows 2000.

Please give Sphinx-4 0.1 alpha a try and post your questions, comments, and feedback to one of the CMU Sphinx Forums:

https://sourceforge.net/forum/?group_id=1904

We can also be reached at cmusphinx-contacts@lists.sourceforge.net.
Chapter 2

Introduction

Author: Arthur Chan, Rita Singh, Editor: Arthur Chan

In this chapter, we describe the history of Sphinx as a tool of speech recognition and give an overview of different software packages of Sphinx.

2.1 History of Sphinx

2.1.1 1970-1988

Author: Rita Singh and Arthur Chan, Editor: Arthur Chan

Many people will begin the history of modern speech recognition in early 1970s. Before we go on, it will be interesting to understand the historic background of speech recognition at those years.

Nowadays, hidden Markov Model (HMM) is a de-facto standard of speech recognition. At the time 1970s, perhaps only a few can foresee this future. Many of researchers perhaps will even feel that speech recognition is a groomy topic.

In pre-1970, perhaps the most frustrating event for a researcher in speech recognition is the critism of John Pierce from AT&T.

In 1969, Dr. John Robinson Pierce, an eminent scientist of AT&T wrote a 2 page letter to prestigious Journal of Acoustical Signal of America (JASA) to critize the research of speech recognition. The letter was
titled “Withered speech recognition. His belief is that speech recognition research lacks of a clear goal and vigorous scientific method.

This created a devastating effect to the field of speech recognition. For the fact that John Pierce is in a very high level in the corporate ladder of AT&T research. ¹

This letter stimulates a lot of researcher to seriously think of the feasibility of practical speech recognition. A speech recognition working group, led by Professor Allen Newell, carry out a three years research from 1970 to 1973 to study the feasibility of speech recognition. The Newell’s report include the study of implementation of the speech recognition system Hearsay.

This is the time, the idea of creating the first CMU’s HMM-based recognizer is conceived by Dr James Baker, a graduate student of Professor Raj Reddy. Many people regarded him as one of the founding fathers of modern speech recognition. (Another one is Professor Federick Jelinek, now in John Hopkins. )

HMM at that time, is “an alternative method for automatic speech recognition”. It is an alternative method because it is purely statistical and the model is always assume to be probabilistic. The HMM models sound units by representing it as a graph of states and where each state has its own distributions. The model parameters including parameters for distribution and the transition weights. of the graph is solely learned from data. The decoding of the waveform to speech is done by Viterbi algorithm, already widely used in convolutional code decoding. The computation required by Viterbi algorithm is fairly intensive because all possible hypothesis specified by the HMM will be evaluated and the amount of these hypothesis will exponentially increase with the length of the utterances.

At the beginning, although HMM-based speech recognition represented a paradigm shift from the signal and rule-based approaches followed by almost all researches at that time. It was believed to be too computationally intensive. Although one important speed-up technique, beam pruning is demonstrated in the recognizer. The notion of speaker-independent speech

¹J. Pierce is also the father of Communications satellites. He was the inventor of the Pierce Gun, a vacuum tube that transmits electrons and is used in satellites. He is also the one coined the term “transistor”. In 1969. In AT&T lab, he oversaw work on mathematics, statistics, speech, hearing, behavioral science, electronics, radio waves and guided waves. Therefore, his word makes the funding for researching speech recognition suddenly decrease in 1969-1975. This also made speech research in AT&T lab stop for a while.
recognition system still doesn’t make sense to many people.

During the 80s, several research teams, notably IBM and AT&T start to apply statistical approach to small domain problems such as speaker dependent systems or speaker-independent isolated word systems.

True speaker-independent recognition of continuously spoken speech still presented a problem. By the late 1980s, researchers were beginning to discover ways to deal with them, it was still generally felt that the computational resources of the time, which consisted of slow processors and tiny memory by today’s standards, could simply not support an HMM-based speaker-independent, continuous speech recognition system. In 1988, CMU has carried out influential research which incorporated several new techniques in the modeling of spoken sounds and the engineering of the actual algorithms used for recognition. It is possible to build a better speaker-independent system. This was a continuous-speech speaker-independent system that not only recognized speech with high accuracy, but did so at the natural speed at which words are spoken by an average person (real-time recognition). The system was developed by Kai-Fu Lee, then a doctoral student under the supervision of Professor Raj Reddy at CMU, and Roberto Bisiani, a research scientist at CMU.

Since 1988 until the present day, as computers and algorithms have both grown in sophistication, Sphinx has morphed into a suite of recognition systems, each marking a milestone in HMM-based speech recognition technology. All of these retain the name of Sphinx, and are labeled with different version numbers, in keeping with contemporary style of referencing software. In the paragraphs that follow, we will describe some key technical aspects of these systems.

2.1.2 1988-2003

Author: Rita Singh, Editor: Arthur Chan

[Editor Notes: I copied everything from Rita’s web page about history]

[of Sphinx. I added only a few things at the end of Sphinx III and]

[Sphinx IV. I believe the material is good enough.]

There are currently four versions of Sphinx in existence:
Sphinx I

Sphinx I was written in the C programming language and was, as described in the paragraphs above, the world’s first high performance speaker-independent continuous speech recognition system. It was based on the then viable technology of discrete HMMs, i.e. HMMs that used discrete distributions or simple histograms to model the distributions of the measurements of speech sounds. Since speech itself is a signal that can take a set of values that are continuous in some range, the modeling paradigm required a quantization of speech into a discrete set of symbols. Sphinx I accomplished this using a vector quantization algorithm. A primary sequence of LPC-cepstral vectors was derived from the speech signals, and from this sequence, two secondary sequences of difference parameter vectors were derived. The vector quantization algorithm computed codebooks from the vectors in the sequences, and replaced each vector by codeword indices from the codebooks. HMMs were then trained with these sequences of quantized vectors. During recognition, incoming speech was also converted into sequences of codeword indices using the same codebooks.

The sound units that the system modeled with discrete HMMs were called generalized triphones. A triphone is simply a distinct phonetic unit labeled with its immediately adjacent phonetic contexts. Triphones were, and remain, one of the most effective innovations in modeling speech sounds in HMM-based systems. In a system based on generalized triphones, a number of triphones are modeled by a common HMM. In Sphinx I, this was done for logistical reasons, since computers in those days were not powerful enough to handle all triphones separately. Also, the large stored databases required to train the vast repository of triphones found in normal everyday speech did not exist.

In a recognition system, the actual process of recognition is guided by a grammar or language model, that encapsulates prior knowledge about the structure of the language. Sphinx I used a simple word-pair grammar, that indicates which word pairs are permitted in the language and which are not.

Sphinx I achieved word recognition accuracies of about 901000-word vocabulary tasks. Such performance represented a major breakthrough in those times. The system performed in real time on the best machines of the time, such as the SUN-3 and DEC-3000.
**Sphinx II**

Sphinx I triggered a phase of phenomenal development in HMM-based continuous speech recognition technology. Within five years of its introduction, technology based on semi-continuous HMMs was ready to be used, and much of it had been developed at CMU. Sphinx II came into existence in 1992, and was again a pioneer system based on the new technology of semi-continuous HMMs. The system was developed by Xuedong Huang at CMU, then a post-doctoral researcher working with Professor Raj Reddy. Like Sphinx I, it was written in the C programming language.

The essence of semi-continuous HMM based technology was that speech was no longer required to be modeled as a sequence of quantized vectors. State distributions of a semi-continuous HMM were modeled by mixtures of Gaussian densities. Rather than the speech vectors themselves, it was the parameters of the Gaussian densities that were quantized. Sphinx II used 4 parallel feature streams, three of which were secondary streams derived from a primary stream of 13-dimensional cepstral vectors computed from the speech signal. All components of the feature streams were permitted to take any real value. For each feature stream, Gaussian density parameters were allowed to take one of 256 values (the number 256 being dictated by the largest number representable by an 8-bit number). The actual values of the 256 sets of parameters were themselves learned during training.

In order to achieve real-time speeds, the system was hardwired to use 5-state Bakis topology HMMs for all sound units. Each sound unit was a triphone, but unlike Sphinx I, this system did not use generalized triphones. Instead, state distributions of the HMMs for the triphones were tied, i.e. states of the HMMs for several triphones were constrained to have the same distribution. The state tying was performed using decision trees. This technique of sharing distribution parameters at the state level, invented by Mei-Yuh Hwang, then a doctoral student at CMU under the supervision of Professor Raj Reddy, was yet another major milestone in HMM-based speech recognition technology, as it made it possible to train large numbers of parameters for a recognition system with relatively modest amounts of data.

Sphinx II also improved upon its predecessor Sphinx I in being able to use statistical N-gram language models during search, where N could be any number, in principle. An N-gram language model represents the probability of any word in the language, given the N-1 prior words in a sentence, and is significantly superior to the word-pair grammar used by Sphinx I as a representation of the structure of a language.

The semi-continuous HMM based Sphinx II required much greater com-
putation to perform recognition than Sphinx I did, and consequently the early Sphinx II decoders (the part of the recognizer that actually performs the recognition) were relatively slow, and took longer than real time on the hardware of the day. These were soon replaced by the FBS-8 decoder, written by Mosur Ravishankar of CMU (Ravishankar, 1996), which used a lexical-tree-based search strategy that represents the recognizer vocabulary as a tree of phonemes. This lextree decoder could perform recognition in real time on standard computers. The name FBS-8 recalls an interesting history of this decoder. FBS-8 stands for Fast Beam Search version 8. This decoder was actually the 8th in a string of quite different decoders. In fact its most successful predecessor, FBS-6, used a "flat" search strategy that represents each word in the recognizer’s lexicon separately, and thus was coded very differently from its immediate successor FBS-7, which was lextree based. Sphinx II was able to achieve an accuracy of about 9030,000 word vocabulary (e.g. Wall Street Journal) recognition task.

**Sphinx III**

The next milestone in recognition technology was marked by the creation of Sphinx III [2]. It was developed four years after Sphinx II was unveiled, and incorporated technology that allowed the modeling of speech with continuous density HMMs in a fully continuous vector space. No vector space quantizations were required at any level. Naturally, this resulted in better recognition performance as compared to its predecessor, Sphinx II. However, at the same time due to statistical requirements imposed by this technology, large amounts of data were required to train such HMMs well. Fortunately, hardware improvements in computers allowed the storage and processing of large amounts of data by this time. Sphinx III was thus a very viable system at the time it was developed. It had many new features, but was at the same time designed to be backwardly compatible with Sphinx II. This meant that it could handle both semi-continuous and fully-continuous HMMs. The system was written in the C programming language, and developed jointly by two people at CMU: its HMM-training modules were developed by Eric Thayer, and its decoding, modules were developed by Mosur Ravishankar. Sphinx III was more versatile in its feature-type handling capacity than its predecessors. The primary feature stream was no longer constrained to be 13 dimensional. It could use single and 4-stream feature sets. The HMM topology was user-specifiable. Like its predecessors, it used triphone HMMs, with state sharing information obtained using decision trees.

Sphinx III currently has two different decoders, both written by Mosur Ravishankar. Both decoders use statistical N-gram language models
with \( N_l=3 \) during search. They differ in several respects, including their search strategies. One decoder, generally referred to as S3.0, uses the flat search strategy. The second decoder has two variants, usually referred to as S3.2 and S3.3. These differ in their ability to handle streaming input and return partial hypotheses during decoding. S3.2 does not have these capabilities while S3.3 does. Both variants use a lextree based search strategy. The use of lextrees makes these versions much faster than S3.0, and their speed is further enhanced through a subvector quantization strategy for Gaussian selection. Both decoders, and the trainer, have hardwired requirements to some degree, though to a much lesser extent than the Sphinx II system. Such requirements are often an essential part of the engineering that goes into implementing speech recognition systems, and are necessitated by the limitations of the computational platforms of the day.

**Sphinx IV**

The years after the development of Sphinx III saw two major advancements in speech research. First, multimodal speech recognition came into existence, with the development of associated algorithms. Second, it became feasible for speech recognition systems to be deployed pervasively in wide-ranging, even mobile, environments.

In multimodal speech recognition, the information in the speech signal is augmented by evidence from other concurrent phenomena, such as gestures, expressions, etc. Although these information streams are related, they differ both in the manner in which they must be modeled and the range of contexts that they capture. The contexts that affect the evidence for any sound can often be highly asymmetric and time dependent. It therefore becomes necessary for the structure and the distributions of the HMMs, and the context dependencies of the sounds they represent, to be variable and user controllable. Also, wide ranging practical applications require the system to function in widely varying language scenarios, and so it also becomes necessary to be able to use language models other than statistical N-grams, such as context free grammars (CFGs), finite state automata (FSAs), or stochastic FSAs, which might be more optimal for the task domain at hand. Statistical language models are just one of a set of different types of well-researched models of human language.

With these considerations, by the beginning of the year 2000, it was clear that Sphinx III would need to be upgraded. Sphinx III could only use triphone context sound units and required a uniform HMM topology for all sound units. It also required a uniform type of HMM, in the sense that the basic type of statistical density used by each HMM would have to be the
same, with the same number of modes, for all sound units, regardless of
the amount of training data or the type of feature being used for recogni-
tion. Sphinx III, moreover was only partially ready for multimodal speech
recognition - although it could use multiple feature streams, it could com-
bine them only at the state level.

Additionally, in the time since the development of Sphinx III, the world
had seen an exponential growth and innovation in computational resources,
both hardware and software, available to the general user. Internet, markup
languages, programming languages which augment and integrate smoothly
with markup languages and the Internet etc. have developed rapidly. Even
better recognition algorithms are now available, and internet based tech-
nology permits new ways in which these can be integrated into any real
medium. High flexibility and high modularity are the norms by which
today’s pervasive software is being developed.

Thus, in response to the demands and offers of new technology avail-
able to speech recognition systems, and to the demands of the rapidly
maturing area of multimodal recognition, Sphinx IV was initiated by a
team of researchers in 2001, joining forces from Carnegie Mellon Univer-
sity, Mitsubishi Electric Research Labs and SUN Microsystems. In 2002,
researchers from Hewlett Packard Inc. joined the effort. At the time of
writing this article, Sphinx IV is nearing completion. It is written entirely
in the JAVA programming language, which is an extremely powerful lan-
guage for Internet based applications. It provides vastly superior inter-
facing capabilities. The system is being developed on an open platform,
and is available to researchers, developers, and commercial organizations
freely at any stage of development. It is in principle an effort of the world
community.

2.1.3 From 2003-2005 June

Author: Arthur Chan, Editor: Arthur Chan

As at this manual is written, there are still three parallel lines of effort in
developing the Sphinx Speech Recognition System. We will give a brief
summary of this effort.

Sphinx II

Motivated by project LISTEN lead by Prof. Jack Mostow, a new FSM based
speech recognizers was created by Ravi Mosur. Dr Mosur also added new
functionality such that continuous HMM is supported by Sphinx II. This
occurred in version Sphinx 2.5 of the recognizer. Sphinx 2 is regarded as one of the future candidates of open source embedded speech recognition. It is now widely used by developers and researchers. Several companies have modified Sphinx 2 to create commercial speech recognition products.

**Sphinx III**

Motivated by project CALO (Cognitive Agent that can learn and organize) which required a highly accurate speech recognition engine with real-time speed. Facility of fast GMM Computation was incorporated by Arthur Chan and Evandro Gouvea to speed up the speed recognizer.

Project CALO also requires intelligence of component that could learn, part of the effort, mainly carried out by David Huggins-Daines and Arthur Chan. Maximum likelihood linear regression (MLLR) was incorporated in source code of SphinxTrain and Sphinx III. This consistently results in 5-10% of relative improvement of speech recognition accuracy.

Another major improvement is the availability of live mode decoder APIs written by Yitao Sun. This provides Sphinx II-like application interface such that developers can call and create application. Combine it with Sphinx III highly accurate speech recognition, satisfactory applications were easily created.

Part of the effort in Sphinx III is also to re-structure and re-organize the source code of s3fast and s3slow to make them coexist with each other. In Sphinx 3.5, this effort is completed such that the two recognizers now exist in the same code base and reduce in large amount of code duplication. The effort of this part will continuously improve the code base such that a more efficient working environment could be created for advanced users and developers.

**Sphinx IV**

A beta release of Sphinx IV is completed. Sphinx IV developers, notably from Sun Microsystem, continue to add facilities such as confidence measure into Sphinx IV. Sphinx IV allows developers to program in an well-structured programming language, JAVA. It supports multiple types of knowledge source that includes n-gram language model and finite state machine. Multiple developers contribution has enrich the code base such that it become more versatile and useful.

Effort is still continued by Willie Walker from Sun Microsystem to create a acoustic model trainer in Java. This effort will lead to increase in
productivity of researcher of acoustic modeling.

2.1.4 Summary

This section summarizes the effort of developing Sphinx. If you are interested to learn more about history of Sphinx or the effort of the current development, please visit the following web sites:

- CMU Sphinx Home page
  
  http://www.cmuSphinx.org

- Dr. Rita Singh’s web page
  
  http://www-2.cs.cmu.edu/~rsingh/homepage/Sphinx_history.html

- Arthur Chan Sphinx Developer’s web page
  
  http://www-2.cs.cmu.edu/~archan

2.2 SphinxTrain

Author: Arthur Chan, Editor: Arthur Chan

A good decoder is just half of the success story of a good speech recognition system. A good speech recognition system requires an acoustic model which could model speaker characteristics. Therefore the availability of a trainer becomes very important for development of open source speech recognition.

At the age when Sphinx I and Sphinx II is developed and widely used internally CMU. Dr. Eric Thayer, a research scientist, in CMU first grouped and created the prototype of current SphinxTrain and was widely used at the time 1997-1998 within CMU. The scope of SphinxTrain contains several types of tools.

- **Parameter Estimation:** For example, **bw** is a tool that carry 1 iteration of Baum-Welch training given a set of waveforms.

- **Data Conversion:** For example, **mk_s2** family of tools are used to convert Sphinx III models to Sphinx II models.

- **Utility** For example, **cp_parm** is used to copy parameters from 1 set of HMM to other. **wave2feat** converts waveforms to mel-frequency cepstral coefficients (MFCC).
The development of the trainer and Sphinx III decoder was carried out at the same time. The two programs were developed separately with predefined common file formats. SphinxTrain fullfill the need of training two types of models that could be used by Sphinx II/ Sphinx III and Sphinx IV’s recognizer. They are semi-continuous HMM (SCHMM) and fully continuous HMM (FCHMM). It is widely employed in CMU for training of acoustic models for multiple corpora.

A lot of further refinement of SphinxTrain was carried out by Dr. Rita Singh with a lot of talented students and researchers of CMU Robust Group. For example, the current version of decision tree-based state-tying for continuous HMM was realized and refined by Dr Singh. She also proposed an novel way to automatically create linguistic questions for decision-tree-base state tying.

There are several notably improvements from 2001 to 2005

1. Ansification of code Dr. Alan Black and Mr. Kevin Lenzo carried out a lot of code refinement for SphinxTrain (such as ansification.) before the current source code is released as an open source project at 2001.

2. Perl wrapper To enhance the process of training, a set of perl of scripts is incorporated into SphinxTrain. It is contributed by Ricky Houghton.

3. Adaptation facilities two tools, mllr_solve and mllr_transform is incorporated by David Huggins-Daines and Arthur Chan. They are largely based on the Sam-Joo Doh ‘s speaker adaptation tool kit which is used internally in CMU.

2.3 A brief Historical Note of CMU-Cambridge LM Toolkit

The version 1 of the toolkit (was called CMU LM toolkit) is created by Roni Rosenfield. Later Philip Clarkson in Cambridge University improve its caching capability and data structure so that it can handle more data with less memory. It becomes version 2 of the program and is open-sourced since 1996. Version 2 is a significant re-write of Version 1 which significantly improve code efficiency and memory usage. It is widely used by researchers in different research fields.
2.4 Why do I want to use Sphinx?

If you are a developer, the major reason you want to use Sphinx is because Sphinx currently is the most complete archive of speech recognition tool. Sphinx is also released using a liberal license. You may aware that other very powerful toolkits, are also open-sourced. Unfortunately, some of their licenses disallows you to use their code in your project. It may also disallow you to use the code and distribute them later in your project.

Sphinx allows you to have full control of your source code as well Sphinx's code. You are free to use, modify and distribute your code with Sphinx's code. In another way, Sphinx is not using GPL, that means you don't have to make the code to be free and keep it close source. We believe this is the kind of flexibility developers really want.

Another aspects of Sphinx is that based on Sphinx and its related resources, and with enough knowledge 2, you can build a lot of different types of applications. By speech applications, I mean things like dictation, automatic speech server. More importantly, the most common denominator or all these applications, i.e the recognizer is open-sourced in CMU Sphinx. You will also get a lot of bundling resource such as dictionary, phone list from CMU's speech archive. This is something very important if you want to have full-control of you speech recognition system.

If you are researchers and your interest is in HMM-based speech recognition, there are several reasons why you may want to use Sphinx III and Sphinx IV as your research tools.

First of all Sphinx assumes that GMM computation and the search process are separate. The important consequence is that you could carry out two different types of research without conflicting each others. For example, you could try to device a new observation probability without touching the source code of the search routine. At the same time, you could also build a new search algorithms without thinking of the GMM computation. This gives you a better advantage in speech recognition research.

The second reason is in training. Most of the time when you were doing modeling research, what you would like to change is the algorithm of estimation. SphinxTrain’s Baum-Welch’s algorithm solve this problem in two stages. It first dump the posterior probabilities count in a separate file and can be easily readable by libraries of SphinxTrain. You can just play with these counts and there is no need to device you own Baum-Welch trainer. This advantage can greatly shorten your research time from weeks to dates.

2The knowledge of proper usage of a speech recognizer and produce high performance results will be as difficult as modifying the gcc compiler or the linux kernel source code.
The third reason is in the code-clairity, later recognizers such as Sphinx III and Sphinx IV have put readability of the code as one of the most important design goal. Many researchers, not only they want to use the recognizer as a tool. They also want to change the code to suit there purpose. Sphinx III, for example, has only 4 layers of code from main() to the GMM computation. This become a good advantage for researchers who like to make changes in a speech recognizer.

2.5 The Hieroglyphs

Author: Arthur Chan, Editor: Arthur Chan

That is this manual.

As a continuous effort of improving every aspect of Sphinx, one of the goal of the team is to create high quality documentation for expert as well as general users. At 2004, this manual was created by Arthur Chan to fulfill this goal.

The project name “Hieroglyph” symbolize two aspects of documentation of speech recognizer. First of all, the usage a speech recognizer and building sucessful speech recognition system required large amount of knowledge. The depth of the knowledge required by a researcher can be compared with how a type of character could ended. The second aspect is that building a speech recognizer is sometimes very hard and that might required the developer or the researcher. It sometimes requires to pay effort that is similar to learn a type of Hieroglyphs.

The first draft of the Hieroglyph document aims at provide a comprehensive guide on how to use Sphinx III, SphinxTrain and CMU LM Toolkit to build a speech recognition system. Future plan will include the use of Sphinx II and Sphinx IV.

As at 2005 Jun, 70% of the first draft of the hieroglyph document is completed and currently could be found in

http://www-2.cs.cmu.edu/~archan/sphinxDoc.html

2.6 The Sphinx Developers

The Sphinx Developers are the people who maintain and continue to develop aspects of Sphinx’s project. A large portion of them are from aca-
demic institutes such as CMU and MIT. There are also significant amount of them are from companies such as Sun, Compaq and Mitsubishi.

The development of CMU Sphinx is not exclusive to Sphinx Developers. Users are welcomed to contribute patches of code to Sphinx and patches are continuously incorporate to enrich and strengthen the current code base. If you are interested in developing Sphinx, you could contact the maintainers: Arthur Chan, Evandro Gouvêa and Alan Black.

2.7 Which Sphinx(en) should I use?

It depends on what kind of application you are trying to build. Here are couple of consideration you need to think about.

- **FSG and natural language** In one word, if you expect to build a system with highly structured dialog. Possibly an FSG system is suitable for you. Currently, your choice will be either Sphinx II or Sphinx IV.

  If you want to build a system with more natural input and less constraints. Then a speech recognizer + robust parser system will be something more suitable to you. What you need to do is to use a speech recognizer to generate output string. Then use a parser to parse this possibly errorful output. The later can be well handled by parser such as Phoenix.

- **Programming Languages** Sphinx II and III is in C which is quite portable in general. Sphinx IV is in Java. You may also need to decide whether you and your team can handle either of these two languages. Many people are religious in only one programming language. The different versions could give you a lot of flexibility to choose.

- **Accuracy and Speed**

  We didn’t carry out very extensive benchmarking of different Sphinxen. The following can only be regarded as a rough estimate According to our measurement. The accuracy numbers can be roughly summarized as

  $$\text{Sphinx IV} \geq \text{Sphinx III} \gg \text{Sphinx II}$$

  and in terms of speed,
Sphinx IV >= Sphinx III >= Sphinx II

That means Sphinx II is still the fastest recognizer we had in our inventory. However its accuracy is also the lowest. Sphinx III contains our best C’s recognizer which can be configured as both fast and pretty accurate. Sphinx IV is possibly the most all round recognizer we have and contrary to people’s belief, it can actually be pretty fast. Partially, Java’s development has come to a stage where it can handle heavy loaded computation.

- **Interfaces**

  Sphinx have several versions. Depends on your goal, your choice of recognizer will be different.

  In general, Sphinx IV provides the best interface among all. It can be used by specifying a config.xml and command-line. This advantage can make your web developers are much easier task in general.

  Usage of Sphinx II and Sphinx III requires much more skill in scripting and in general understanding of the program. At the current stage, they are stil regarded as systems for expert users. Though the Sphinx team is seeking to continuously improve the interface and try to make it easier to use for more users.

- **Platforms**

  Sphinx II, III and IV are all highly platform-independent. However, the use of Java language in Sphinx IV could potentially allow higher degree of platform independency.

  It is also of common interest that whether any version of Sphinx, as at Aug, 2005. David Huggins-Daines created an embedded version (based on Sharp Zaurus) for Sphinx II. In that case, Sphinx II will perhaps the choise of the recognizer for embedded platform.

- **Research** We would recommend you to use Sphinx III or Sphinx IV because both of them have clean design. They all supports continuous HMMs which is currently a de-facto standard of HMM

  In the case of doing individual research. We have the following recommendation.

  For acoustic modeling and fast GMM computation , Sphinx IV is actually are pretty good research platforms for speech recognition. By itself, it support SCHMM, CHMM and Sub-vector quantized HMM. They represent there different kinds of modeling technique for speech recognitions.
For research in search, Sphinx IV is a good candidate, because you can avoid annoying memory management and focus on implementation of the search algorithm.

For speaker adaptation or general acoustic model research, you will find that you need to directly change SphinxTrain. The stage's estimation process will be a very nice process for you to use.

It is very unlucky that currently Sphinx's architecture makes feature extraction research pretty difficult to carry out. In future, we will try to change situation.

- Documentation, support and community

As at 2005, all three Sphinxen (II, III and IV) are actively maintained by current maintainers of CMU Sphinx. Most of the developers have years of experience in large scale code development and industrial experience. They usually provide very fast turn around to most of the problems.

Documentation in both Sphinx III and Sphinx IV are probably more comprehensive than that of Sphinx II. Both Sphinx III and Sphinx IV provided Javadoc style of code documentation as well very detailed documentation for the software itself.

For the community, one could imagine Sphinx IV attracts more young, energetic researchers. Whereas, many users of Sphinx II and III turns out to be experts with experience. Though as each of the software turns out to be more balanced, more users found that all three recognizers are pretty good choice in general.

### 2.8 Hephaestus

Author: *Arthur Chan*, Editor: *Arthur Chan*

Project Hephaestus of CMU is a project that tries to collect open source toolkits in speech and language processing. CMU Sphinx is one of the key tool in this project. A lot of effort is also put in toolkit such as speech synthesis, dialogue system and general speech resource. In this section, you will learn some of these efforts. They are important because just having the speech recognizer cannot build interesting application such as dialog system. Running a recognizer and a trainer also requires many relevant resource such as dictionary and training data. Therefore, it is very important to have some basic understanding of each of these components.
2.8.1 Festival

Festival is a free open-sourced text-to-speech synthesizer developed in Edinburgh University. It is now mainly maintained by Prof. Alan Black in CMU. In the simplest case, you can feed in a string and festival will create the waveform and output through your speaker. The functionalities of festival allow users to modify speech-synthesis in multiple levels. It is also extremely flexible and has very good programming interface.

Festival has very good documentation on-line at

- Festival's main page [www.cstr.ed.ac.uk/projects/festival/](http://www.cstr.ed.ac.uk/projects/festival/)
- FestVox [www.festival.org](http://www.festival.org)

2.8.2 CMU Communicator Toolkit

CMU Communicator is a project that allows users to get flight information using speech. It is led by Prof. Alex Rudnicky in CMU. The whole system is available as a tool kit so the public can used it freely. CMU Communicator contains several of open source components including Sphinx (the recognizer), Phoenix (the robust parser), Helios (post-processor that generate confidence), Rosetta (the language generation routine) and Festival (speech synthesizer). Therefore, this is one very interesting demonstration on how speech recognition system can be built in general.

Another important aspect of it is that CMU Communicator data is free to be used and its collection process allows you to use is data as freely as you like. CMU will allow charge you the distribution charge for the whole acquisition process.

For more detail about Communicator, you can go to

[http://www.speech.cs.cmu.edu/Communicator/](http://www.speech.cs.cmu.edu/Communicator/)

For the information of the corpus, you can go to

[http://www.speech.cs.cmu.edu/Communicator/Corpus/](http://www.speech.cs.cmu.edu/Communicator/Corpus/)

for details.

2.8.3 CMUdict, CMUlex

CMUdict is the dictionary collected and built in CMU from its 15 years of research. CMUdict is the short-form of Carnegie Mellon University Pro-
nouncing Dictionary. You can download it at

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

### 2.8.4 Speech Databases

CMU provides few databases for research purpose. For example, AN4 and Communicator databases. Please contact the maintainers: Arthur Chan, Evandro Gouvêa and Alan Black for further detail.

### 2.8.5 OpenVXML and OpenSALT browser

Voice XML (VXML) and Speech Application Language Tags (SALT) are two markup languages that integrates speech services into markup languages. It becomes a very important toolkit to build speech application for people who used to web development. You can find their information at

http://www.speech.cs.cmu.edu/openvxml/index.html

and

http://hap.speech.cs.cmu.edu/salt/

### 2.8.6 Miscellaneous

CMU’s speech group’s web page contains a lot of useful information for using, building and researching speech recognition systems. You can find its web site at

http://www.speech.cs.cmu.edu/

### 2.9 How do I get more help?

CMU Sphinx is currently maintained by CMU, a lot of developers/researchers from research institutions such as Sun Microsystems, Hewlett-Packard, Mitsubishi and MIT. Feel free to ask any questions in the discussion forum at

http://sourceforge.net/forum/?group_id=1904
Chapter 3

Introduction to Speech Recognition

Author: *Arthur Chan*, Editor: *Arthur Chan*

There is a general need for accessible introductory texts for modern speech recognition. Most of the text found in the web assume readers to have certain level of knowledge in probability and mathematics. The following text attempts to guide the reader through a step by step guide to understand the basics of the theory of speech recognition. For more in-depth discussion, please consult the reference we quoted at the end of this chapter.

The way of the presentation is greatly influenced by the style of Dr. Kevin Knight’s “A statistical MT Tutorial Workbook” so you would probably see a lot of places where the word “sinister” appears. That essentially means some fancy terms that many technical people talk about but could be understood pretty easily if you know what it actually means.

Important terms will be underlined throughout the presentation.

3.1 The big picture

This guide will try to give an accessible presentation that is similar to chapter 6 of “Speech recognition” (Right name?) by Rabiner and Juang. We will also material such as basic probability theory for the readers.
A general disclaimer, even with this guide, it is still up to the Sphinx users or reader patience to complete this tutorial and gain the essential understanding. This will be quite mathematical. That is to say, you will see funny symbols all over the places. However, the author of this guide do believe if one could finish this tutorial, they would have as much as knowledge as most “speech experts” in the field!

In general, we would hope every Sphinx users to learn these basics before they start to use either the recognizer and the trainer. Not only because they are survival skills if one would like to build a successful speech application, \(^1\) but also they are very interesting (i.e. why we wrote about it).

Enjoy!

### 3.2 Basic probability

In this tutorial, you will see a lot of funny symbols look like this \( P(X) \). This looks fairly sinister if you don’t know what it means. Let us speak in English, \( P(X) \) just means the probability that \( X \) happens. This is actually what most people talk about the word “chance”. For example, one would say “there is half a chance to have head when you flip a coin”. A fancy way to write it is

\[
P(H) = \frac{1}{2}
\]

A refined way to read it is “the probability of getting the event \( H \) will be one half” which we defined \( H \) to be “the event of getting head when flipping a coin”. You could also read it as “how likely a head will appear when flipping a coin”. So if someone tell you \( P(H) \) for one coin \( A \) is 0.9 and for the other coin \( B \) is 0.5, you will know that flipping coin \( A \) will give you a better chance to got a head.

There are some other terms you would always hear from people. Here are what they actually mean.

- \( P(X) \), a probability. The chance that \( X \) happens. Depends on the situation, it will also be called prior probability.

\(^1\)Yet, they were rarely properly taught.
• $P(X|Y)$, a conditional probability. That is to say the probability of event $X$ happens given the event $Y$ happens. Using the coin A/coin B example above, then we could say, $P(H|B) = 0.9$

Another useful example is more related to speech recognition. Let’s say when I have a waveform $O$ and we want to know likely it is “Hi, my name is George Bush.” or let us denote this sentence as $W$. Then the probability will be $P(W|O)$.

• $P(X,Y)$, a joint probability. That is to say the probability of both event $X$ and $Y$ happen together.

Here comes a big question. Why do we bother with the business of condition probability in this first place? What is the difference between it and the probability itself? The answer is in some situation, another event may not affect how likely something will happen. For example, let me ask you this, how likely will you got a head if you toss an unbiased coin, if your previous 100 trials of tossing are all tails? The answer is still one half. That is to say

$$P(X_{101}) = P(X_{101}|X_{100} \ldots X_1)$$

This is a situation call independence between events. Or another way to say it if $X$ and $Y$ are independent, then $P(X|Y) = P(X)$

Another thing which is good to know is the relation between joint probability and conditional probability.

In general,

$$P(X,Y) = P(X|Y)P(Y)$$

In the case when the two events are independent then, we can even write an even nice expression.

$$P(X,Y) = P(X)P(Y)$$

Now let’s say we want to know the probability of getting two ones in two dice tossings would be simply $\frac{1}{6} \times \frac{1}{6}$. That is $\frac{1}{36}$.

There are other funny properties of probabilty, for example, it has to be smaller or equal to 1 and larger or equal to 0. i.e.

$$0 \leq P(X) \leq 1$$
also, if there are only two possible events X, Y, then sum of their probabilities will be equal to 1

\[ P(X) + P(Y) = 1 \]

These funny properties, from my point of view is more a mathematical convenience than a must. However, the advantage of such a setting will actually give us an easier time when we try to deal with problems later.

**Exercise 1** \( P(X, Y) = P(X|Y) \times ? \)

**Exercise 2** What is the chances of getting all head if tossing a coin for 100 times?

### 3.3 What speech recognizers actually do?

How a speech recognizer works? One mystery for beginner would be how a speech recognizer recognize the things I am talking? To understand this, let us look in this way. When you speak to a computer, your speech will be transmitted as an acoustic energy and the information will be encoded in the acoustic waveforms. The receiver of the sound signal (aka microphone) pick this signal up and digitize it to machine readable form which is now a binary file. Then, the speech recognizer will be asked, what does this waveform mean?

Talking about “meaning” can end up with lengthy and useless philosophical discussion. So let us see how engineers define this problem, they end-up to design the speech recognizer to decide what is the best possible word sequence that represents the waveform.

What does that mean? That is to say the speech recognizer would enumerate all sentence it “knows” and try to match them with the waveform one by one. That’s it.

**Exercise 3** The grammar of a computer speech recognizer only include one word, “Hello”. If you say something to the recognizer, what answer will it give to you?

---

2that’s what the sinister term “best word sequence” means, if you have a sequence of words, it is just a sentence.
The last exercise is not a trick question. The answer is simply “Hello”! So you see one important job for the engineer is to design a good recognizer is to design the vocabulary. The best speech recognizer (in a technical sense) would give the same dumb answer if the person who used it hasn’t use it correctly. ³.

Let us have a closer look of a speech recognition system in figure. Figure 3.3. It is also the basic architecture of how many speech recognizer was based on. We have the input as an acoustic waveform and we will try to decide what the best word sequence is. We also mentioned the part where the computer knows a set of predefined word sequence, a fancy way to call is a knowledge base. You will see later that actually consists of multiple different things.

Let us try to write down what we want to do in mathematics. As we say, what we want is to enumerate all possible word sequences. Let us call this set to be $W^*$ and each individual word sequence is $W$. Then our task is to compute the conditional probability of each of them given the waveforms, $0$.

\[
P(W_1|0) \\
P(W_2|0) \\
P(W_3|0) \\
\ldots
\]

³There is another possible answer for the above question, that is the recognizer could decide that nothing should be recognized for the utterance. It would be too much to describe how a recognizer in this tutorial. We will probably do it in later chapters.
\[ P(W_n | 0) \]

So what we want is to find the word sequence that gives \( \max_{W \in W^*} P(W|O) \). So we denote that as \( \arg \max_{W \in W^*} P(W|O) \).

Many people will tell you the formulation is called source-channel formulation and one could treat the acoustic signal as the source and the process of hearing is the channel. Information theory is very good at dealing with this kind of system and can give a lot of insights. For us though, thinking in this way will give us one convenience. That is we will not mixed the order of the \( W \) and \( O \) in the conditional probability formulation (coz the input is always the condition).

### 3.4 Bayes Rule

There is one rule you should memorize in speech recognition and I think that would be Bayes Rules

\[
P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}
\]

It is pretty easy to prove.

**Exercise 4** Prove Bayes Rule. (Hint: You could probably make use of formulae mentioned in previous sections)

Why it is important? Mathematically it is nice to know this kind of reciprocal relationship exists between condition probabilities \( P(X|Y) \) and \( P(Y|X) \). Sometimes, depends on situation, you could only learn one quantity instead of others. Such a conversion therefore is quite crucial. (This part is a little bit weak.) In the case of our problem, the quantity,

\[ P(W|O) \]

is actually not that easy to model. Therefore what people do was to use Bayes rules to revert the probability.

\[
P(W|O) = \frac{P(O|W)P(W)}{P(O)}
\]

So, the problem of speech recognition will be equivalent to

\[
\arg \max_{W \in W^*} P(W|O) \tag{3.1}
\]

\[
\arg \max_{W \in W^*} \frac{P(O|W)P(W)}{P(O)} \tag{3.2}
\]
At this point, people will start to ignore the term $P(O)$ because we are essentially doing comparison for many terms. $P(O|W_i)P(W_i)/P(O)$ where $W_i \in W^*$. Hence in such a case $P(O)$ will be common term in all comparison and therefore we can just ignore it.
3.5 Basic of Statistics

[We need to talk about the concept of mean, variance and distribution]
3.6 Problems of Speech Recognition

Essentially, the study of speech recognition in these days are the studies of how to accurately estimate and efficiently compute the terms $P(O|W)$ or so-called acoustic model and $P(W)$ or so-called language model. Then, at the end, the decision are merely made by comparing each word sequence and decide which one is the most likely. It is a fairly simple perspective.

Of course, the term acoustic model and language model are fairly sinister if you don’t know what they mean. Before we go on to a formal mathematical treatment. Let us just think about how these two symbols mean.

$P(W)$ is the prior of the word sequence. Why is it important? Let us consider this example. If you already knew that the user will only speak in digits, then in English, it is reasonable to assume there will be only 11 possible words you need to recognize. So, it will be reasonable to say we should put to prior of word sequence that consists only digits to be 1 while others to be 0. In this case, $P(W)$ is essential to constrain the decision process. So that nothing outrageous will be recognized. That is perhaps why people call it the language model.

The above example is an over-simplification of the actual situation. In general, we speak of all types of words all the time. They are not constrained to digits. However, we should recognize the fact that we only speak of a subset of all the words most of the time. For example, let us think of a funny word such as “Gypsy”. Will you say “Gypsy” all the time? Will you say the word “Gypsy” more than the word “I”? Probably won’t. So it is quite appropiate to use probability to describe whether a word will be said in the language model.

The language model, from my opinion is the most important key to build successful speech application. At the end of this tutorial, we will just give a brief check of how it is done and used in practice. We will use Chapter 8 to describe more on that part.

Let us turn into another funny entity $P(O|W)$. Essentially, what it models is when a word is said, how likely the waveform would looks like $O$. That is why usually people call it the acoustic model. What does it mean for a computer?

There is a pretty standard practice of how a word could be represented. For a word like “Gypsy”, we know that it is pronounced as “Jeep see”. Recognizers such as Sphinx use the phone based representation such as “JH IH P S IY” to represent the word. There is a special term of basic

---

4. one, two, three, four, five, six, seven, eight, nine, zero, oh.

5. This is the so-called CMU phoneme set. BTW, a lot of people don’t really like it. So
unit that represent pronunciation and we called it phones, some people will call it phonemes as well. Phone is one type of sub-word unit.

Usually, whenever a word is to be recognized, then the recognizer will look up a table that shows the mapping of word to phone sequence, this mapping, in layman term is just the dictionary.

We still have to ask how a phone unit could be modeled. This is a longer story. Now let us back track a little bit on looking at the speech signal itself. We will use the next two sections to talk about it.

**Exercise 5** Try to do the basic derivation from the posterior probability $P(W|0)$ to a representation that consists of the two major components of speech recognition.

**Exercise 6** Identify what you will say when are speaking a phone number (in your country) and the date of a year. What are the constraints you will imposed to the speech recognition? If you express the constraints in a graph, how will it look like?

---

6The are different, phoneme is the logical basic non-confusable unit of pronunciation. Phone is their actualy realization.
3.7 Speech Signal in time-domain

Speech is essentially a type of sound. What is sound then? Sound is essentially the vibration of air molecules. In a human vocal tract, speech is caused by vibration of vocal cord. This will make the sound we could hear. Our capability of speech and language are probably benefited by our vocal tract which can change its shape to create different sound.

The first thing you got to know about speech is that its production process is not very exact. For example, did you try to record your speech and show it to yourself? This is a lot of fun and I always do it myself. Let us try out an experiment. Let us recorded two “hello”s, speaking just

I have recorded two “Hello”s myself as in Figure 3.7 and Figure 3.7.

As you could see, even the content is the same “hello”, the waveform could be significantly different. Though, one thing we are sure, although the waveforms are not exactly the same, they are definitely similar.

For many programmers, this is perhaps something very hard to grasp because most of the time their jobs are deterministic in nature. Human’s faculty of speech is also so advanced that we treat it as something as mechanical and deterministic as well. Though, there is an easy way to convince yourself. That is to do the above experiments once. One software we would recommend is KTH Wavesurfer

http://www.speech.kth.se/wavesurfer/

It is an open source software and it is very easy to use in recording.
Exercise 7 Try to record your voice using wavesurfer and observe the waveforms.

This “randomness” of speech is very hard to deal with it. In general, the “hello” you say this time will be different from the “hello” you say next time. The “hello” you say will definitely different from the “hello” other people said at any time. This explains the intrinsic difficulty of speech recognition.

Now, let us turn around, can we say there is nothing similar between the two “hello” we recorded? We also couldn’t say that. Using the waveforms and try to enlarge it in a finer scale, you could already come up with such as speculation. However, time signal has too much fluctuations and it is very hard to get useful information. Therefore, one simple way is to analyse the waveform based on its frequency components.
3.8 Speech Signal as shown in a Spectrogram

In general, one could always think of a waveform as composed of superposition of several sinusoids. One could also always decompose (FIX ME! except in some rare conditions?) a waveform to its components through a technique called Fourier Analysis.

We will do one step further because usually speech signal’s characteristic will change from phone to phone. However, generally people found that speech signal could be is generally stationary, or precisely speaking, its statistical properties will not change. A signal that has such a property a usually called short-time stationary.

What we will do is something like this. First, we will first break down a whole waveform to small blocks. Say each block will have 0.01s of speech. We call each block to be one frame. Then, we carry out fourier analysis on each block. This will allow us to see how strong is each frequency components and each frame. Now imagine, if we plot the magnitude of the frequency component against both time-frequency, it will then get a 3-D graph that describe that the change of frequency components in a speech signal. People used a nice way to do so by making something called spectrogram. Instead of

Now follow the previous example, I created the spectrograms of my two “hello”s. Both of them are generated using the “spectorgram” functionality of wavesurfer. A note on how to read the spectrogram. From left to right, you could see the magnitude of frequency component of each frame. The darker the spot, the stronger the magnitude of the the component.

From the spectrograms of two “Hello”s, you might be able to observe that there are more similarities of them. As a matter of fact, for the same word, a spectrogram usually look similar even it is spoken by different speakers. When we think about it, it is just because a certain sound are usually pronounced by changing the shape of the vocal track in a similar way.

Exercise 8 Try to generate the spectrograms of the two waveforms. Do they look similar?

7It seems to me this one step took people 30 years to make. We should probably talk more on it

8For experts, the spectrograms might have already shown that my pronounciations are sloppy and I have accents. Please forgive me, this is a more tutorial than an detailed discussion of spectrogram reading.
Researchers used to propose spectrogram reading as a means for learning English for deaf people. This was, however, not that successful as the researchers wish for. It turns out that even though spectrogram could look pretty similar, it requires a lot of learning and experience to derive the original words just from the spectrogram alone. Therefore, in this tutorial, we skip the detail of spectrogram reading and ask the readers to look up the references. (say [] and []). We probably won’t see them too much in this book, spectrogram reading is indeed a very useful skill and could provide a lot of insights in practice.

Let us summarize what we have seen in the last two sections.

- Apparently, cutting the waveform into small parts and analyse them gives a clearer picture to a speech waveform. As a matter of fact, this idea, short-time Fourier analysis will be one thing that we will always do in future. (in Sphinx as well)

This also give a slightly different point of view of a waveform, from now on, instead of treating it a sequence of PCM value, we should probably treat it a sequence of frames. This point of view is crucial to understand how search algorithm works in speech recognition.

- It matters a lot of how you represent each frame. If you think back how we use samples to represent a frame, you will find that using Fourier analysis to represent each frame makes much more sense to many.

One could call this representation issue as coding issue, as we try to represent a bunch of numbers with another. In Sphinx and in many
recognizers, a type of feature called mel-frequency cepstral coefficients (MFCC) is always used. We will not talk what MFCC is in this tutorial. You could find the details in Chapter 6.

**Exercise 9** Just a thinking problem and a warm up exercise of the next section. Try to think about how do you compare two waveforms. We would give some possible answers in the next few sections.
We always come up with a situation where we need to measure whether things look alike or similar. The first question we should ask is What is similar? We always say two things are similar without asking ourselves why. Obviously, we cannot expect computer can judge without doing any measurement. So, our first question to ask is how similar of two numbers? And how we could use computer to measure the similarities.

Let us start from comparing two numbers, how similar are two numbers? Let us not ask any trick question here. One intuitive solution is quite simple. Consider numbers $a$ and $b$, then the smaller their differences, $a - b$ can be a measure of similarity. Now there are still some complications here. Are we talking about difference of their magnitudes? or just there difference? Do we care about the difference is positive or negative? Lets say we exercise our imagination a bit. Then we can write down

$$a - b$$

Their difference

$$|a - b|$$ The absolute value of their difference

$$|a| - |b|$$ Their magnitudes difference

$$||a| - |b||$$ The absolute value of their difference of magnitudes

If you are more mathematically inclined, you will say that $|a - b|$ is the correct one. I bring this up because in my point of view usually there is no correctness in some issues in real-life.

Let us seek for an intuitive measure of distance is Euclidean distance. For a 2-dimensional case, for two points $(a_1, a_2)$ and $(b_1, b_2)$, their distance could be defined as

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

$|a - b|$ happens to be also a special case of Euclidean distance when in one dimension. Therefore, it is quite intuitive to use the absolute difference in that case.

We could extend the above two dimensional case to n-dimensional where Euclidean distance could be defined as

$$\sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

One problem of Euclidean distance is that it assumes difference of each components to have equal weighting. So for example, if we assume any person could be represented by two dimensions, length of palm and weight of the person. Then, the weight factor will become dominating at the end.

Therefore, in general, it makes sense to weight the distance accordingly.
\[
\sqrt{\sum_{i=1}^{N} w_i (x_i - y_i)^2}
\]

How to choose the weight then? This is a tough problem. One measure called Mahalanobus distance used the variance of the sample in each dimension to weight the distance. For example, if the variance of a particular dimension is smaller.

\[
\sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}}
\]

If you take a look of the form, it resembles to the exponents of what you see in the Gaussian distribution. This is a special form when one assumes components does not correlate with each other.

**Exercise 10** Could you write the form of Mahalanobus distance such that the correlation of components are also considered?

### 3.9.1 Comparing Two Time Sequences

Both numbers (or scalers) and vectors are pretty easy to deal with. It becomes non-trivial when we need to compare two time sequence. It could be a time sequence of scalers or it could be a time sequence of vectors. Simply because the length of the sequence could be different.

This problem becomes very important when we need to compare one waveform to another waveform. Before that let us consider a simpler but similar question, how similar are sequence “a b c d” and “a b c”. Does “a b c d” and “a c” more similar than “a b c d” and “a b c”?

Probably we should define what is similar first before we made the problem become subjective. Let us just assume we have a way to align two different sequence together.

For example, we could align “a b c d” and “a b c” like this.

\[
\begin{array}{cccc}
a & b & c & d \\
a & b & c & -
\end{array}
\]

Then we will find that sequence “a b c” have deleted one alphabet. In general, the number of insertion/deletion/substitution in this case turns out to be only 1.

while for “a b c d” and “a c”, we would have an alignment like this

\[
\begin{array}{cccc}
a & b & c & d \\
a & - & c & -
\end{array}
\]
Then, in this case, sequence “a c” will delete two alphabets. So probably we could say “a b c” and “a b c d” is more similar in this case.

Let us ask ourselves this. How could we come up with this alignment in the first place?

What we could do is that we could try aligning the two sequences “a b c” by exhaustively to “a b c d”. Say, we could try the following alignments:

\[
\begin{align*}
\text{a b c d} \\
\text{a b c -}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a b - c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a b -- c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a - b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a - b - c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a - - b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a -- b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a -- b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a -- b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a - a b c}
\end{align*}
\]

or

\[
\begin{align*}
\text{a b c d} \\
\text{a - a - b c}
\end{align*}
\]
or
a b c d
- - a b c
or
a b c d
- - a b c
or
a b c d
- - a b c
or
a b c d
- - - a b c
or
a b c d
- - - - a b c

After all this effort, we could then see which way of alignment looked more “aligned”. (In this case, the first alignment turns to have the least amount of insertion, deletion and substitution) However, as might have observed, the number of combinations to enumerate could be increased exponentially.

[Editor Notes: Also need to add the exact analytical results]

It is therefore important to device a more efficient way to obtain such an alignment. The idea is called **dynamic programming** and will be introduced in the next section.

### 3.9.2 Dynamic Programming

The expansion of search space is called in the literature [1], the **curse of dimensionality**. The basic concept of dynamic programming is ??.

[Editor Notes: Still need to give better description on dynamic programming at here. Also we need to try to explain the basic setting.

---

9The same term also appear in literature of pattern recognition, but it has different meaning.
At the end, the string matching was essentially just a graph search problem. We could illustrate in the graph in Figure 3.9.2.

![Search Graph](image)

**Figure 3.5:** Search Graph for matching “a b c d” and “a b c”. The dark line represents the final alignment.

Sequence matching is a very simple example of dynamic programming. In this chapter, we will meet several examples where the concept of dynamic programming will be used. The three main problems of HMM-based speech recognizer used the concept of dynamic programming.
3.10 Hidden Markov Model

3.11 Why people used HMM?

3.12 The three problems of HMM

3.13 Problem 1: The most likely path problem

3.14 Problem 2: The total likelihood problem

3.15 Problem 3: The parameter estimation problem or training problem

3.16 Practical acoustic modeling

It is important to point out that our discussion on the three problems are only basics of HMM. If you just follow the description, the result will probably be not accurate enough and not fast enough for real-life usage.

Throughout this book, several advanced techniques will be presented. Some are fast decoding technique and some are acoustic modeling techniques. Each of them are very interesting on its own. We will just point you to individual chapters for further reading.

- Decision Tree Tying
- Speaker Adaptation
- Fast Decoding
3.17 Some basics in language modeling

3.18 Sphinx as a Toolkit of Building Speech Recognition System

After going through this whole chapter, I believe you have certain understanding about speech recognition. What is Sphinx's position in speech recognition then?

Sphinx is the crystal of research for more than fifty researchers and graduate students in last 20 years. It consists of the recognizer, Sphinx and the trainer SphinxTrain. So, it could help to solve the computation problem and the estimation problem of HMM.

The strength of Sphinx also lies on its several algorithms of doing large vocabulary speech recognition. For example, starting from Sphinx 3.6, there are several search modes in Sphinx that allow different types of operation such as phoneme recognition, force-alignment could be done. Specialized optimized search model for Finite state grammar, flat and tree lexicon n-gram are also provided.

The rest of this book will discuss various aspects of Sphinx and its related tools. This section is more an introduction to concepts of speech recognition. To start to build a system, we recommend you to read chapter 5. This will give you a concrete idea on how one could use Sphinx to create a speech system quickly.
Part II

Tutorial
Chapter 4

Software Installation

Author: Arthur Chan, Editor: Arthur Chan  This chapter describe general installation procedure for multiple software toolkits that you will need to build a speech recognition system. We will assume Unix is the environment we will use. If you were a window user, we will recommend you to use cygwin. 

There are couple of components you need to download before we start. Make sure you have the following standard software that can be found in Unix. These are tools if you installed linux or cygwin in a windows machine. They usually come standard. They are also not so difficult to get.

- **tar and gzip** Software to decompress a file. If you are Windows users, WinZip, WinRar and 7-Zip will work for you.

- **gcc** C compiler, Sphinx 3 supports both gcc 3.2.2 and 2.95 to know the version, type gcc -v

- **make** A utility that automatically determine which part of the software need to be compiled given a makefile.

### 4.1 Sphinx 3.X

Thee first software we will get is Sphinx 3.X and you can download it at [http://sourceforge.net/projects/cmusphinx/](http://sourceforge.net/projects/cmusphinx/)
Go to the section of “Latest File Releases” and click sphinx3. Then choose the top most labelled with sphinx3-0.5.tgz is the one. ¹

We will go through the standard compilation Linux toolkit compilation process here (i.e. configure → make → make install).

1. **Configure the makefile**
   
   $ configure

2. **Compile the program**
   
   $ make

3. **Install the program**
   
   $ make install

   By default the program will be installed in /usr/local/bin/, if you want to change the path of installation, you can type
   
   $ configure--prefix=<yourpath>

   in the configuring step.

### 4.1.1 SphinxTrain

If you want to get a copy of SphinxTrain, you need to use cvs to get it. Again, you can use the above configure→make→make install process to compile the code.

1. **Check out the source code** You could get the latest source code from Sourceforge by typing
   
   $ cvs -d:pserver:anonymous@cvs.sf.net:/cvsroot/cmusphinx co SphinxTrain

   When you were ask for password, just type enter. Then check out process will start.

   You will see something like this on the screen
   
   cvs checkout: Updating SphinxTrain
   
   U SphinxTrain/COPYING

   ¹For some reason that caused by early interaction of sphinx developers and Sourceforge, sphinx3-0.X.tgz end up means sphinx3.X. It becomes a legacy problem that later maintainer just follow the old rules.
2. **Configure the makefile**

   $ configure

3. **Compile the program**

   $ make

### 4.1.2 CMU LM Toolkit

You can get the Toolkit at

   http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html

1. **Configure the makefile**
   
   > configure

2. **Compile the program**

   > make

### 4.1.3 Other tools related to the recognizer

There are two extra tools which are not packaged in the above distributions. One is called cepview which can allow one to inspect the mfcc file. One is called lm3g2dmp which can transform standard text-based LM to
a specific DMP file format that could be accepted by Sphinx 2 and Sphinx 3. Both of these could be found in the share directory in Sourceforge. You could get the latest source code from Sourceforge by typing

$ cvs co -d:pserver:anonymous@cvs.sf.net:/cvsroot/cmusphinx co share

4.2 FAQ of the installation process

4.2.1 What if I am a user of Microsoft Windows?

Our recommendation for you is that you should consider to use cygwin.

What if I know only how to use Visual C++? So in that case I won’t be able to use Sphinx? Don’t worry. Sphinx’s developers have create a setup for Microsoft Visual C++. From our testing, it works for both VC 6.0 and VC.Net at 2003. Thought, with the constant change of Microsoft’s changes of programmers user interface. It is very unlikely we can follow all the changes they made. So if you have a new compiler, you may need to make the setup itself.

For sphinx 3.X, you just need to go to click program.dsw and do a Build. Then you are done. The files can be found in ./bin/{Debug,Release}.

For SphinxTrain, you also just need to go to click program.dsw and do a build. The files can be found in ./bin/{Debug,Release}.

Very unfortunately, there is no VC setup for CMU LM Toolkit. May be we will try to fix this problem later.

4.2.2 How about other Windows compilers?

The software itself doesn’t impose any restrictions on the compilers. The problems where most people are facing usually caused by lack of knowledge of the compiler itself.

The basic principle of compilation is simple. First create libraries for libaudio/, libss3decoder/ and libutil/. Then use the compile the files in program with the libraries. The VC setup are basically done by the above procedure.

If you work out a setup to compile Sphinx, please kindly contact the maintainers: Arthur Chan, Evandro Gouvea and Alan Black.
4.2.3 How about other Unix platforms?

Sphinx 3 and SphinxTrain are written in ANSI C and they are fairly portable to other Unix platforms. Though there is not much testing has been done for platform other than windows, Linux and Mac OS X. Though, we continuously try to make sure the code is as portable as possible.

4.2.4 What if I can’t build Sphinx 3.X or SphinxTrain?

Please report to the Sphinx’s Developers.

4.2.5 What if I am not a fan of C?

In that case, try Sphinx 4, it is a better, cleaner system written by Java. Documentation of Sphinx 4 is not the purpose of this manual. You can check out its installation procedure at

http://cmusphinx.sourceforge.net/sphinx4/

4.2.6 How do I get help for installation?

Try the Sphinx Forum in Sourceforge, there are multiple experts there to help you. The URL is

http://sourceforge.net/forum/?group_id=1904
Chapter 5

Roadmap of building speech recognition system

Author: Arthur Chan, Editor: Arthur Chan

Sphinx 3 and SphinxTrain are good candidates for application programmer to build a speech recognition systems. By itself, it only provide a bare bone demonstration system (or livedecode) for speech recognition. This is far from what most user need. \(^1\) This chapter provides a step by step guide to build a speech recognition system.

There are two approaches of building a speech recognition system.

The first way is to create a system by using the decoder of Sphinx and some open source models of Sphinx, this mode of development is easiest.

The knowledge required in building a speech recognition system using the first approach can be divided into two parts. One is to understand the specialty of using speech as an input of a program. For basic knowledge of HMM-based speech recognition, please consult Chapter 3 for further detail. The other part is to interface the speech recognizer with other programs. Both of this knowledge is not trivial. The first 5 steps will guide you through the procedure.

The second way and a prefered way to create a system is to build a full system including the acoustic model yourself. There are several reasons for doing so. One of them is despite the fact that the model you could found in open source model archive is fairly good. They may not be the

\(^1\)Unfortunately, many users have misunderstanding that livedecode is a full system.
best for *your* situation. For example, if one would like to build a simple digit recognition system, models trained with multiple hours of data may have Gaussian distribution which is too "flat" that doesn't match with the special need of digit recognition. Training your own model can allow you to control the quality of the acoustic model. \(^2\).

Experts of the second mode of development usually attain advanced knowledge in pattern recognition, thorough understanding in speech recognition and also savvy skills of manipulate different tools of speech recognition tools. It is one of the most satisfactory process in the world. Step 6 to Step 11 will briefly go through this process. In Chapter 6, a detail description of every sub-component of the process will be described.

After you tried the last two modes of development, you might be interested in 1) tuning the current system to make faster and more accurate or 2) understand what’s going on internal to Sphinx. Step 12 and Step 13 will describe it further.

A final note of this tutorial, there are several high quality tutorials already exists in the web. If you found that this tutorial alone couldn’t solve all of your problems. Also try the following Evandro Gouvêa’s tutorial

http://www-2.cs.cmu.edu/ robust/Tutorial/opensource.html

It consists of a full run-through tutorial which contains a full recipe of training using the AN4 or RM1 database. The resulting system would be a very simple evaluation system for these databases. You are highly recommended to run-through either this tutorial or the on-line tutorial if you are new to speech recognition or Sphinx.

## 5.1 Step 1: Design of a Speech Recognition System

In this section, I will list some concerns you might need to consider before you build a speech recognition system.

### 5.1.1 Do you really need speech?

Not every type of information is suitable for speech input. For example, digit string, is very good candidate for input by speech and it is not very

\(^2\)Sometimes it might give you a job as well.
natural to be input by other modes of input (such as touch pad).

What kind of application is suitable for speech? For example, dictation is well-known to be one application that speech is doing very well. Other applications can be simply summarized as variants of command-and-control type of applications. For example, if you just need several types of “speech shortcut” when you are using Windows. You might just want to have a few commands in your system and when each command was recognized, corresponding action would be taken by your system.

5.1.2 Is Your Requirement Feasible?

Using a HMM-based speech recognition, there are several types of input are very hard to be recognized. Some of them are intrinsic to the problem itself.

Let’s first talk about some linguistic confusion, in general, if there is no linguistic constraints (e.g. N-gram LM or FSM), the recognition rate will usually very low. This situation is a very common and that occurred in research of conversational speech recognition and there is no easy solution to solve it. So if your goal is to recognize everything that can be spoken by anyone, Sphinx probably can’t help you so.

There are two major factors in designing your systems, one is linguistic confusibility and one is acoustic confusability. They are usually the cause of how speech system is poor in practice.

Linguistic Confusibility

If you rank order the difficulties of building a system. You could probably get a list. \(^3\)

1. Recognize everything that could be said by anyone in the world

The case we just mentioned. This is very hard. There are also effects of multiple languages and it is generally not easy to do.

\(^3\)A very important note should be put here. Despite the author put 3000 words to be doable, this is just reflecting an expert level that beginning users could do. It is very important to realize, Sphinx, with one capable speech recognition expert can create arbitrary type of speech application with large amount of vocabulary (> 20k).
2. Recognize everything that could be said by everyone who speaks the same language

   This is slightly easier but still we have no constraints.

3. Recognize what two persons are saying in a conversation

   This might be easier if we could impose a constraint by the topic. For example, if we know that the two persons are programmers and if we know they are likely to talk about programming all the time, then we could probably assume terms such as “program”, “function” could appear more.

4. Recognize what a person want to write in a letter

   This is also known as dictation. This is still quite hard but it is quite constrained because given a topic of the letter and the type of letters in that topic. For example, if you know you recognizer “Dear” in a marketing letter, usually, it will always follow by “customer”. Usually, the amount of words a person could use could be confined to less 60k words. That explain the success of a lot of commercial speech recognizers.

   Sphinx, by itself, does not forbid developers to build a dictation system. However, building dictation system usually required a lot of application tuning and training. This might be too hard to be described in this manual. Just to collect data may take you months to complete. This might not be suitable for this tutorial. As you might know, there are already several sucessfull commercial recognizers in the world. There are also several efforts tried to build an open source speech recognizer using Sphinx. Unfortunately, most of them are following incorrect approaches and using incorrect method. We could only wish someday we would write another guide on how to use Sphinx in dictation.

   After this point, things will start to be more easily doable by you. (Of course if you read this manual.)

5. Build a 3000 word dialogue system

   This is already down and proved to be very sucessful. For example CMU Communicator system is one good example. In the Communicator system

   http://www.speech.cs.cmu.edu/Communicator/

   One could ask information for the itinery of flight. The destination and arrival location can be obtained. This system is very sucessful
because the speech of this type is very constrained.  

Flight information is just one type of information. What the user might want to query could be things like weather information, TV listing, product information, etc. Most of the time, this query is very constrained. This will make your life easier.

6. Build a 1000-person person phone directory system with English names

This is also known *speech directory system* and this is definitely doable and useful. I give this example because it could actually be harder than the task of building a 3000 word dialog system because it reflect the intrinsic difficulty of this problem, sometimes people name sounds very similar. Even human being has problems with them, we will talk about this more when discuss acoustic confusability. If there are more names to be recognized, the chances of names being confused will be higher. This is the nature of the problem. However, this is a problem which is doable by Sphinx and could be done by people who follow this manual.

7. Build a 100-command command and control system in English

This is obviously another easy task, Sphinx could definitely help you. For example, take a look of the “Robosapien Dancing Machine” project.

http://www.robodance.com/

Command and control system is very general term for many type of application. You might want to use it for controlling your windows applications or you might want to use it to control a character in a game.

There are two notes we need to mentioned at this point. The first note is that it is important for you to realize there are different difficulties of building speech recognition system. Some of them is very hard to be tackled by a single person. You might need to form a team of people and try to deal with them with expert guidance. This will set your expectation to a correct level.

The second reason is that we want to show you size of vocabulary is not the only reason why a task is hard. Whether the tasks are difficult partially depends on the perplexity of the language model, a quantity one would describe more in Chapter 9. Another factor is whether words confused

---

Notice that CMU Communicator was only using Sphinx 2, the Sphinx 3 system was found to be 30% more accurate in later years.
with each other acoustically. This is briefly mentioned in the task “Build a 1000-person person phone directory system with English names” Let us go on with several other tasks in the next paragraph.

After you read this part, I urge you to test yourself this. Is recognition of an unconstrained 16 digit string harder? Or is recognition of a credit card (VISA or Mastercard) harder?  

**Acoustic Confusibility**

Have you been to China? And use Chinese romanized name to call your friends? If you have done it once, you will know that it is very hard. Let us consider another task, “Build a 1000 romanized Chinese name directory system”. This task could be extremely hard because romanized Chinese names can be easily confusible with other. Consider my romanized Chinese name:

Chan Yu Chung

and one my friend’s name

Cheng Yue Chun.

Try to read it aloud on the phone, you will know it might be quite impossible even for human ear to recognize them.

The same situation happens in alphabet recognition. It is generally very difficult to differentiate between some of them. For example, A, J and K could hard to differentiate. B, C, D, E, G and V, Z could be hard to differentiate. M and N is yet another pair.

Yet another example of acoustic confusability is so called phoneme recognition. For example on the phone, many human misrecognized the word “Arthur” to be “Althur”. This is a case where the system should consider to reprompt the user for spelling or other means of confirmation.

Another possibility is even more subtle. For example, it might happened that the two things you like to recognize have the same name. For example, there are two people who has the same name. This is a natural situation where name cannot be the unique identifier for them. Obviously, a speech recognizer could not do too much about that.

Acoustic confusibility is one important concern to build a system. If you put in a lot of confusible words, it will end up to give you a very poor recognition rate.

---

5The second case is easier because credit card number are usually generated by some special algorithm and can check whether the 16 digits were valid or not.
5.1.3 System Concern

Running a speech recognizer could take a lot of computation of your machine. The speed of the system mainly depends on the size of the vocabulary of the task and the number of mixture component in the Gaussian distribution (A model parameter, detail can be found in Chapter 6).

We will recommend you to have a PC which has comparable performance with a Pentium 1GHz or more to use Sphinx 3. The recommended system is a Pentium 2GHz or more. This is a range where a lot of the system (<= 10 K Vocabularies) can be run under 1 times real time. That is to say, a 1 second of speech will be recognized using less than 1 second of computation time. However, a task with only 10-100 words will have no problem in running under 1 times real time if a 500MHz machine is used.

5.1.4 Our recommendation

We don’t recommend first-time user to build system more than 10k because at that limit expert choice is required to tune the system down to 1 times real time. This estimates is valid at Jun 2005. The member of the Sphinx team alway continues to improve the speed of the performance. May be when you read this manual later, the speed of the recognizer will be fast enough for your system requirement.

A vocabulary which is too large has another problems, most of the time such a system is not suitable for the user’s requirement. This recommendation will hopefully make sure that the user will consider their requirement carefully.

5.1.5 Summary

Design your speech recognition system is very important. Not everything is suitable to be input by speech. There are also some requirement of vocabulary could be impossible to implement. Some of them might not be possible even when human is a listener.

Speed is another concern that one might need to consider and they are usually related to the size of vocabulary.
5.2 Step 2: Design the Vocabulary

5.2.1 The System for This Tutorial

In this tutorial we will just a recognition system that allow users to recognize 5 key phrases, “spread sheet”, “word processor”, “calculator”, “web browser” and “movie player”. The user would can say these 5 words and the system will control the system and start the corresponding applications. This is a very simple system but we could learn a lot of interesting aspect of speech recognition design. We will continue to develop this simple system first. Later we will then expand this simple system.

5.2.2 Pronunciations of the Words

Let’s look at the words we chose again.

spread sheet  
word processor  
calculator  
web browser  
movie player

There are several interesting aspects for the choice, the first aspect is that the words have at least 2 or more syllables. This is chosen because speech recognizer could have hard time to recognizer keywords.

Another aspect is that the word is not easy to confuse with each others. One simple common sense test is try to read pairs of word in pair. This is not automatic method but it is a basic principle of how people will figure out confusability of English words.

As you already learnt in Chapter 4, Sphinx is a phone-based speech recognizer. Simply speaking, in the run time, the recognizer will create a giant Hidden Markov Model (HMM) and the search will try to search the best path given this HMM. Now we have the word list we want to recognize. We obviously need the phone-based representation of these words. Some people will just call this representation as “pronouciations”.

How do we come up with these pronunciations? This is actually quite simple. What happened if need the pronunciation of a word? The obvious answer is to look up this pronunciation from the dictionary.
Create the Phonemes Manually

So the first way to get this pronunciation is to lookup a dictionary to decide this phone set. The caveat of doing this is that you need to make sure phoneme symbols you were using is the same as the phoneme set used in the open source acoustic model.  

Another concern is that the dictionary you were using might not match the dialect of English (same for other languages). For example, British English, American English and Australian English are actually quite different. If you used a British English dictionary for American English, the end result may not be the most ideal one.

In our case, let us simplify our discussion by using a simple choice of dictionary. If you are using open source acoustic model that can be found, the following method might be easier. You could use free dictionary such as cmudict and you can download it at

[http://www.speech.cs.cmu.edu/cgi-bin/cmudict](http://www.speech.cs.cmu.edu/cgi-bin/cmudict)

CMUdict is the short term of Carnegie Mellon University Pronouncing Dictionary. It is a machine-readable pronunciation dictionary for North American English that contains over 125,000 words and their transcriptions. Though the pronunciation is mainly for American English, it is actually a reasonable start.

The current version is CMUdict 0.6 and if you look inside this file (you could use the command cat to do it).

```
,COMMA K AA M AH
-DASH D AE SH
-HYPHEN HH AY F AH N
...ELLIPSIS IH L IH P S IH S
.DECIMAL D EH S AH M AH L
.DOT D AA T
.FULL-STOP F UH L S T AA P
.PERIOD P IH R IY AH D
.POINT P OY N T
/SLASH S L AE SH
```

---

6If they were not matched, Sphinx’s decode and decode_anytopo will inform the user (you) that they couldn’t find the model. So the recognizer will be stopped in the initialization.
The first column at every entries is the word, the rest is the phoneme representation of the word.

What does this phoneme means? The following is a table that shows their usage.

We might talk more about the pronunciation table. The phoneme example table is useful when you start to change the pronunciations yourself.
**Usage of automatic tools**

After having cmudict, you can write a simple perl script to look up the the pronunciation for each word. This is actually pretty simple so we skip how to do it at here. I will recommend you to try to do it because it is fun.

Many people has gone through this process and some of them are nice enough to share these resources. For example, one of them, Prof Alex Rudnicky and Mr. Kevin Lenzo from CMU have written a web page.

[http://www.speech.cs.cmu.edu/cgi-bin/cmudict](http://www.speech.cs.cmu.edu/cgi-bin/cmudict)

(The same web page we used in downloading cmudict v0.6) You could just look a pronunciations up by typing the word in the edit box “Look up words or a sentence”.

You will soon get the pronunciation of each word

```
SPREADSHEET S P R EH D SH IY T
WORD_PROCESSOR W ER D P R AA S EH S ER
CALCULATOR K AE L K Y AH L EY T ER
WEB_BROWSER W EH B B R AW Z ER
MOVIEPLAYER M UW V IY P L EY ER
```

We will call this file **simple.dict**.

Yet another way to do it is to use the QUICK LM tool by Prof. Alex Rudnicky.

[http://www.speech.cs.cmu.edu/tools/lmtool.html](http://www.speech.cs.cmu.edu/tools/lmtool.html)

You could first create a file like this

```
SPREADHSEET
WORD_PROCESSOR
CALCULATOR
WEB_BROWSER
MOVIEPLAYER
```

And let us called the file to be **simple.txt**.

You could submit the file just by clicking “Compile Knowledge Base” button. You would see the phone set will be slightly different. You need to map AXR to ER, replaced AX to AH, replaced all TS by T and S, replaced all PD by P, replaced all DX by either D or T. This could be easily done by perl or sed.
Whether you use automatic pronunciation tools or manual way to create the pronunciation. Do remember that you could freely change the dictionary yourself. Be careful though, the best pronunciations are those that best fit for your users but not the official pronunciation of a word. The general phenomenon is that human display a very high varieties of pronunciation. The so called “official” pronunciations may not fit into your cases.

Here comes a question that you will always come up with. If we fiddle the system parameters or settings, how could we know whether these changes give a positive effect to the system? The key is to benchmark the system every time a change is made. Testing the system by actually using it is not a bad idea. However, as HMM-based speech recognition system is statistical. A failure for once, or a failure out of ten times doesn’t tell you too much about the performance of a system. They are just not statistically significant enough to tell you much. Therefore, it is important to do some quantitative benchmarking.

One metric people usually used is WER. If your change improve the WER in a large test set usually means the performance of the system really improves. We will discuss technique of benchmarking and the definition of WER more later in Step 6 of this Chapter.
<table>
<thead>
<tr>
<th>Phoneme</th>
<th>Example</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>odd</td>
<td>AA D</td>
</tr>
<tr>
<td>AE</td>
<td>at</td>
<td>AE T</td>
</tr>
<tr>
<td>AH</td>
<td>hut</td>
<td>HH AH T</td>
</tr>
<tr>
<td>AO</td>
<td>ought</td>
<td>AO T</td>
</tr>
<tr>
<td>AW</td>
<td>cow</td>
<td>K AW</td>
</tr>
<tr>
<td>AY</td>
<td>hide</td>
<td>HH AY D</td>
</tr>
<tr>
<td>B</td>
<td>be</td>
<td>B IY</td>
</tr>
<tr>
<td>CH</td>
<td>cheese</td>
<td>CH IY Z</td>
</tr>
<tr>
<td>D</td>
<td>dee</td>
<td>D IY</td>
</tr>
<tr>
<td>DH</td>
<td>thee</td>
<td>DH IY</td>
</tr>
<tr>
<td>EH</td>
<td>Ed</td>
<td>EH D</td>
</tr>
<tr>
<td>ER</td>
<td>hurt</td>
<td>HH ER T</td>
</tr>
<tr>
<td>EY</td>
<td>ate</td>
<td>EY T</td>
</tr>
<tr>
<td>F</td>
<td>fee</td>
<td>F IY</td>
</tr>
<tr>
<td>G</td>
<td>green</td>
<td>G R IY N</td>
</tr>
<tr>
<td>HH</td>
<td>he</td>
<td>HH IY</td>
</tr>
<tr>
<td>IH</td>
<td>it</td>
<td>IH T</td>
</tr>
<tr>
<td>IY</td>
<td>eat</td>
<td>IY T</td>
</tr>
<tr>
<td>JH</td>
<td>gee</td>
<td>JH IY</td>
</tr>
<tr>
<td>K</td>
<td>key</td>
<td>K IY</td>
</tr>
<tr>
<td>L</td>
<td>lee</td>
<td>L IY</td>
</tr>
<tr>
<td>M</td>
<td>me</td>
<td>M IY</td>
</tr>
<tr>
<td>N</td>
<td>knee</td>
<td>N IY</td>
</tr>
<tr>
<td>NG</td>
<td>ping</td>
<td>P IH NG</td>
</tr>
<tr>
<td>OW</td>
<td>oat</td>
<td>OW T</td>
</tr>
<tr>
<td>Y</td>
<td>toy</td>
<td>T OY</td>
</tr>
<tr>
<td>P</td>
<td>pee</td>
<td>P IY</td>
</tr>
<tr>
<td>R</td>
<td>read</td>
<td>R IY D</td>
</tr>
<tr>
<td>S</td>
<td>sea</td>
<td>S IY</td>
</tr>
<tr>
<td>SH</td>
<td>she</td>
<td>SH IY</td>
</tr>
<tr>
<td>T</td>
<td>tea</td>
<td>T IY</td>
</tr>
<tr>
<td>TH</td>
<td>theta</td>
<td>TH EY T AH</td>
</tr>
<tr>
<td>UH</td>
<td>hood</td>
<td>HH UH D</td>
</tr>
<tr>
<td>UW</td>
<td>two</td>
<td>T UW</td>
</tr>
<tr>
<td>V</td>
<td>vee</td>
<td>V IY</td>
</tr>
<tr>
<td>W</td>
<td>we</td>
<td>W IY</td>
</tr>
<tr>
<td>Y</td>
<td>yield</td>
<td>Y IY L D</td>
</tr>
<tr>
<td>Z</td>
<td>zee</td>
<td>Z IY</td>
</tr>
<tr>
<td>ZH</td>
<td>seizure</td>
<td>S IY ZH ER</td>
</tr>
</tbody>
</table>
5.3 Step 3: Design the Grammar for a Speech Recognition System

Grammar plays an important part of a speech recognition system. It governs how the recognizer will actually recognize in the hypothesis. In speech recognition, there are two prominent ways to represent grammar. One is n-gram. The other is finite state machine. In this tutorial, we will use the n-gram because currently Sphinx 3 only supports N-gram. If you would like to use finite state machine, you might consult the documentation for Sphinx 2 or Sphinx 4.

5.3.1 Coverage of a Grammar

Generally speaking, if the grammar is too constrained, it won’t cover the words which would be said by the users. In another words, a lot of Out-of-Vocabulary (OOV) words will be resulted. One trivial way to solve this problem is to add new words. However, blindly increase the coverage without moderation can easily introduce unnecessary confusion to speech recognition. Therefore, careful decision and benchmarking must be made before adding new words to the grammar.

5.3.2 Mismatch between the Dictionary and the Language Model

One common confusion is how the speech recognizer behave when the vocabulary of words and the grammar does not match. In Sphinx recognizers, both the unigram of the language model and the dictionary will decide whether a word should exist in the search graph of speech recognition. The recognizer will try to look up the pronunciation (or the phoneme representation) of a word from the dictionary. If the recognizer failed to find a the pronunciation of a word, the word will not be included in the search.

In other words, the search graph will include the set of words that is an intersection of the vocabularies of dictionary and the unigram. This is very likely to change because the implementation existed in Sphinx 2 could probably be used in Sphinx 3 as well.

From this point of view, it is actually ok to include a dictionary with large amount
5.3.3 Interpretation of ARPA N-gram format

The first thing if you used an n-gram based speech recognizer is the ARPA n-gram format. Hence it is quite important to learn how to interpret the format. The following is one example of it.

[ Editor Notes: This example was not shown correctly. ]

.... Comment Sections Skip here ....

data
ngram 1=2001
ngram 2=29445
ngram 3=75360

1-grams:
-2.0960 <UNK> -0.5198
-2.8716 ’BOUT -0.5978
-4.6672 ’CAUSE -0.3765
-5.1065 ’FRISCO -0.2588
-4.4781 ’HEAD -1.1010
-4.3863 ’KAY -0.1962
-4.5624 ’ROUND -0.5975

2-grams:

3-grams:

end

of words. Though, in the actual implementation, the dictionary is actually read into the memory. Hence the size of the dictionary will be constrained by the system memory the user have.
The data section

This section shows the number of each n-grams. In Sphinx, up to trigram is supported.

The X-gram: section

It specifies the log probabilities and the back-off weights for the N-gram.

The end section

This ends the file.

5.3.4 Grammar for this tutorial

For this tutorial, we just create a very simple language model for our purpose.

```
[ Editor Notes: This file is not shown correctly]

data
ngram 1=107
ngram 2=1
1-grams:
-2.0253 </s> -99.0000
-99.0000 <s> 0.0000
-0.6990 SPREADSHEET 0.0000
-0.6990 WORD_PROCESSOR 0.0000
-0.6990 CALCULATOR 0.0000
-0.6990 WEB_BROWSER 0.0000
-0.6990 MOVIE PLAYER 0.0000
2-grams:
0.0000 <s> </s>
0.0000 SPREADSHEET SPREADSHEET 0.0000
0.0000 SPREADSHEET WORD_PROCESSOR 0.0000
```
At here we use a trick to use n-gram to simulate a grammar where each of the keyword we chose will only be entered once. Because all the possible bigrams are zeroed. The only possible path will the hypothesis which enter a word only once. Let us call this file simple.arpa from now on.
5.3.5 Conversion of the language model to DMP file format

Sphinx 3 actually doesn’t support the text ARPA file format. If you want to use your grammar in Sphinx 3, you could use a tool called `lm3g2dmp` to convert the text file to .DMP file format. You could download it in the `share` directory in CVS from sourceforge using the software installation process discussed in Chapter 4.

Then conversion of the LM could be run as

`lm3g2dmp simple.arpa`

A file named `simple.arpa.DMP` will be created. You will need it in Step 5.
5.4 Step 4: Obtain an Open Source Acoustic Model

Enough for language model, it is time to gather resource such as acoustic model. There are a couple of notes before we start.

The first thing one need to know about open source acoustic model is that they are currently stills some rare and valuable resource in the world. One would perhaps appreciate this fact more when they know how acoustic model training is actually done. One would hope that the availability of open source speech recognizer would probably initiate more effort on building open source acoustic models. Do give two thumbs up to any one who build and open source the model. They actually do a good thing that is similar to St. Teresa for speech recognition.

The second thing about open source acoustic model (as well as language model) is that they might not be always suitable for different people need. In general, using large amount of speech to train a model will make an acoustic model to be generally very accurate. However, in specific domain, say in digit recognition, it might be true that just using utterance with digits will be better in some cases. This explains why a lot of developer found that open source models they could obtain usually don’t satisfy them. If you don’t think the open source models is the best, training is always an option for you. Sphinx also provides a trainer. So you have the freedom to fine-tune the model.

In this section, we will discuss various things one need to do get and use a model. Some basic knowledge on how to interprete the model will also be discussed.

5.4.1 How to get a model?

The first place to try is the open source page.

http://www.speech.cs.cmu.edu/sphinx/models/

There is one model that many people used which is so called the broadcast news model or hub-4 models.

This is one important model you will always come up with. So let us talk about this a little bit.

First of all, the model is trained by CMU and have been trained using 140 hours of 1996 and 1997 hub4 training data. This is probably the largest amount of data which have been put into an open source models.
Another thing that you might notice is that this model is trained for read-speech so this is not particularly good for conversational speech. In general, conversational speech could be indeed quite hard to handle.

The phoneset for which models have been provided is that of the dictionary cmudict_0.6d available through the CMU web pages. This matches with the procedure we were using in Step 2.

The dictionary has been used without stress markers, resulting in 40 phones, including the silence phone, SIL. Adding stress markers degrades performance by about 5 within-word and cross-word triphone HMMs with no skips permitted between states.

There are two sets of models this package, comprised of 4000 and 6000 senones respectively. They are placed in directories named 4000 senones and 6000 senones respectively. The reasons why two sets of models are there partially because the trainer has trained two sets of them such that he/she could test the performance of the two settings. Also partially because we hope you could try which setting is more suitable for you.

There are also two settings you would see, one is 1s_c_d_dd, one is s3_1x39. These funny symbols means the type of dynamic coefficients one would compute. The detail of what they mean could be found in Chapter 6.

The two models could be used in both sphinx 3 fast (decode and sphinx 3 slow decode anytopo).  

---

9At 20050523, this is slightly different from the notes one could found.
5.4.2 Are there other models?

Actually, that are more. For example, the TIDIGITS could be found in ????,
the RM models could be found in ????.

You could also train a model pretty easily for AN4. Though, you could observe those are much small models.

5.4.3 Why Acoustic Models are so Rare?

Acoustic model is actually pretty hard to come up. It takes weeks to prepare and take days to train. Most academic or commercial institutes just keep them because it means one valuable resource for them. Many people appreciate the fact that Sphinx is open sourced. I personally hope that more and more people will realize another thing, the HUB-4 models is open sourced because probably this is the first model one could use it to do large vocabulary speech recognition.

5.4.4 Interpretation of Model Formats

Sphinx’s HMM definition are spread in five different files. They are mean, variance, mixture weight, transition matrices and model definition. It is quite important to have some ideas on what it means before we go on further.

A model definition file

So, let’s start from the model definition file. They are usually located in the model_architecture directory when SphinxTrain’s standard procedure is used to train the model.

Here is an example:

0.3
55 n_base
118004 n_tri
472236 n_state_map
2165 n_tied_state
165 n_tied_ci_state
55 n_tied_tmat
#
# Columns definitions
# base lft rt p attrib tmat ... state id’s ...
+BACKGROUND+ - - - filler 0 0 1 2 N
+BREATH+ - - - filler 1 3 4 5 N
+CLICKS+ - - - filler 2 6 7 8 N
+COUGH+ - - - filler 3 9 10 11 N
+FEED+ - - - filler 4 12 13 14 N
+LAUGH+ - - - filler 5 15 16 17 N
+NOISE+ - - - filler 6 18 19 20 N
+SMACK+ - - - filler 7 21 22 23 N
+UH+ - - - filler 8 24 25 26 N
+UHUH+ - - - filler 9 27 28 29 N
+UM+ - - - filler 10 30 31 32 N
AA - - - n/a 11 33 34 35 N
AE - - - n/a 12 36 37 38 N
AH - - - n/a 13 39 40 41 N
AO - - - n/a 14 42 43 44 N
AW - - - n/a 15 45 46 47 N
AX - - - n/a 16 48 49 50 N
AXR - - - n/a 17 51 52 53 N
AY - - - n/a 18 54 55 56 N
B - - - n/a 19 57 58 59 N
CH - - - n/a 20 60 61 62 N
D - - - n/a 21 63 64 65 N
DH - - - n/a 22 66 67 68 N
DX - - - n/a 23 69 70 71 N
The first few line describes the number of parameters in the whole system

0.3 is the model format version.

55 n_base

n_base is the number of base phone including fillers in the system. In this case, the number is 55.

118004 n_tri

n_tri is the number of triphone in the system. In this case, the number is 118004.

472236 n_state_map

2165 n_tied_state

n_tied_state is the number of triphone in the system. In this case, the number is 118004.

165 n_tied_ci_state

n_tied_ci_state is the number tied CI state in the system. In this case, the number is 165.

55 n_tied_tmat

n_tied_tmat is the number tied transition matrix in the system. In this case, the number is 55.

Now let us try to interpret the following line.

#base lft rt p attrib tmat ... state id’s ...
+BACKGROUND+ - - - filler 0 0 1 2 N

From left to right, it reads the base phone is +BACKGROUND+, no left context and no right context, so it is a CI phone. It is a filler. Its phone ID is 0, its first state has senone ID 0, second state 1, third state 2.
It is quite useful to know how the hmm state would map into a senone. (In HTK terminology, the tied-state.) In the above example, if you know that the senone ID for first state is 0. You just need to look up the file means and variances to get the value of them.

```
AA - - - n/a 11 33 34 35 N
```

Here is another entry in the model definition file. This time, we see a typical CI phone entry. If you used the standard training procedure of SphinxTrain (that you will also have a chance to play with in the second half of this chapter), then all the CI phone HMM's state will have their own senone which is unique for it.

As opposed to that, if you look at a triphone definition.

```
AA AA AA s n/a 11 177 190 216 N
AA AA AE s n/a 11 177 190 216 N
AA AA AH s n/a 11 177 190 216 N
AA AA AO s n/a 11 177 190 216 N
AA AA AW s n/a 11 177 190 216 N
AA AA AX s n/a 11 177 190 216 N
AA AA AXR s n/a 11 177 190 216 N
AA AA AY s n/a 11 177 190 216 N
AA AA B b n/a 11 178 189 219 N
AA AA B s n/a 11 177 189 217 N
AA AA CH s n/a 11 177 189 216 N
AA AA D b n/a 11 178 198 219 N
AA AA D s n/a 11 177 198 214 N
```

You will see multiple phones states are actually mapped to the same senone ID. Why does that happen? This is essentially the result of tying happen in training the context-dependent model. What happen is there are always not enough training data for triphone model. Therefore, it is necessary to cluster the hmm-state to make data could be more efficiently used. Most of the time, clustering are either done by decision tree-base clustering or bottom-up agglomerative clustering.

**The Means and Variances, Mixture Weights and Transition Matrics**

Given the above, you could probably make more sense for the rest of the four files. They are usually located in a directory called *model.parameter*. 
All of them are binary files. To read them, you could use a tool called **printp**.

You could find the detail usage of **printp** in Appendix B. However, as it is the first tool in SphinxTrain, let us make a more detail look on it. Every tool in SphinxTrain are designed in a consistent way such that user could use it easily.

If you just type

```
printp
```

The following help information would be shown (as at 20050811)

- **-help no** Shows the usage of the tool
- **-example no** Shows example of how to use the tool
- **-tmatfn** The transition matrix parameter file name
- **-mixwfn** The mixture weight parameter file name
- **-mixws** Start id of mixing weight subinterval
- **-mixwe** End id of mixing weight subinterval
- **-gaufn** A Gaussian parameter file name (either for means or vars)
- **-gaucntfn** A Gaussian parameter weighted vector file
- **-regmatcntfn** MLLR regression matrix count file
- **-moddeffn** The model definition file
- **-lambdafn** The interpolation weight file
- **-lambdamin 0** Print int. wt. >= this
- **-lambdamax 1** Print int. wt. <= this
- **-norm yes** Print normalized parameters
- **-sigfig 4** Number of significant digits in ’e’ notation

If you type

```
printp -help yes,
```

help information would be displayed.

If you type

```
printp -example yes,
```

Several examples of the command would be shown.

Therefore, if you are familiar with these basic command usages, you could probably have no need to reference the manual too much. All the
symnopsis and examples are all replicated again at the end of this book. This will be a plus for people who like to have a book form of manual.

Let us get back to our example, one practical thing you may want to do from time to time is to inspect whether the mean values make sense. For example, acoustic model training could make some component of means and variance vector to be abnormal.

`printp` will allow you to inspect these values by display them on the screen. For means and variances, you could use

```
printp -gaufn mean,
```

or

```
printp -gaufn var,
```

to show the values. If you want to look at the mean for a particular senone, what you need to do is just to look up the model definition file to know the senone.

`printp` also allows you to load in a model definition so that you don’t need to manually look up the model definition file all the time.

To read mixture weight and transition matrices, you could use the `-mixwfn` and `-tmatfn` in the option list. The rest is pretty trivial, you could probably get it by just running the command once.

### 5.4.5 Summary

In this section, we retrieve a model and learnt how to interpret its parameters. The next section will really use all the resources from Step 1 to Step 4 to build a system.
5.5 Step 5: Develop a speech recognition system

Now you got the acoustic model, language model and dictionary. It is time to use it in the recognizer.

You could think of how people run a speech recognizer in two ways. One is called batch mode, the waveform is recorded first, then decoding is done for the whole waveform. One is called live mode, that is recognition is done on the fly when some speech samples was captured by the audio device.

The two modes are slightly different in practice because usually cepstral mean normalization (CMN) is run before decoding. What CMN does is that it would try to estimate the mean of all cepstrum and use it to normalize all cepstral vectors. (For detail, please read Chapter 6.) In batch mode, the estimation was usually carried out by computing the cepstral mean using all frames. Most of the time, this is usually a very good estimate. In live-mode though, since only part of the utterance is seen, one needs to approximate the CMN process with other means. These approximations usually introduce some degradations.

In this section, we will talk about how we could come up with a system that could that batch mode decoding, live mode simulation and actual live mode decoding. In terms of technical difficulty, you could say the three are in ascending order. So, going through one after one will be a good idea for novices.

5.5.1 About the Sphinx 3’s Package

Before we go on, let us take a look of what one could get from a Sphinx’s package.

Assume you go to the install path as you have specified in Chapter 4, then the following software will be show up in the directory. The following executables are what you could find when you install Sphinx 3.5.

- align allphone dag livepretend
- astar decode decode_anytopo

In windows, you would also another executable called livedecode.

For many, it is not easy to understand what these executables are supposed to do. So here is a simple description. **decode, decode_anytopo.**
livepretend and livedecode are decoders. However, each of them slightly different functions and should be used according to different situations.

- **decode**, also called “s3 fast”, it is a batch mode decoder using tree lexicon, with optimization on both GMM computation and crossword triphone transversal. On accuracy, it is poorer than decode.anytopo (as described below.). On speed, it is around 5 times faster. From Sphinx 3.6, decode also allows decoding using finite state grammar and different implementations of search. Therefore it is a well-balanced recognizer and might be the first choice of batch mode recognition.

- **decode.anytopo**, also called “s3 slow”, it is a batch mode decoder using flat lexicon, with optimization on trigram and cross-word triphones. Its performance is very close to a full trigram, full cross-word recognizer. Internal to CMU, it was used in several evaluation back in 1994 to 2000 and it provides the best performance among all the decoder.

Both decode and decode.anytopo only works with cepstral file but not waveforms. Waveforms need to be transformed by cepstral vectors first using wave2feat.

- **livepretend**, is a live mode simulator using the engine of decode. What happens is livepretend will segment a waveform and feed the segments of speech to the search engine of decode. The segmentation is currently done in an ad-hoc manner by choosing 25 frames to be one segment. As we have explained before, segmentation of speech will make the cepstral mean normalization. So usually, livepretend works slightly poorer than decode.

- **livedecode**, is a live mode demonstration using the engine of decode. livedecode allows the user to try to recognize speech in push-button manner. It is by all means NOT a dictation machine. It is an example to show how the live mode decoding API (as detailed in Chapter 12) could be used to build a system.

After the four major decoding tools, we should look at other tools such as align, allphone, astar, dag. They are search programs with specific purpose.

- **align** Given the transcription of an utterance, align could give the state, phone and word level of alignment to the user.

- **allphone** Carry out phoneme recognition for an utterance with full triphone expansions.
- **dag** Both **decode**, **decode anytopo** could generate lattice for second-stage rescoring. **dag** as one machine which could do this rescoring could search for the best path within the lattice given a trigram language model.

- **astar astar** could generate an N-best list given a lattice. It is very useful for N-best rescoring.

When using all tools we mentioned, it is useful to bear in mind some common design properties of tools within the Sphinx 3.x package. All the tools in Sphinx 3.x has the following options.

- `-logfn`

This could allow one to record every messages of each program.

Applications of Sphinx 3.x could have quite a lot of command line arguments. If you just type the command itself, a fairly detailed command-line usage description could be found.

If you don’t like to specify command line manually, you have couple of options, the trivial one is store the command-line option and put in a Unix script file and it will look like this:

```
decode
-opt1 <val1>
-opt2 <val2>
```

Another way is that you could put the option in one file and retreat it as configuration file. So, you could create a file and look like this:

```
-opt1 <val1>
-opt2 <val2>
```

and called it `config`

Then you just type

```
$ decode config
```

Then decode will automatically use the contents of config to initialize decode. That seem to save a lot of tyings for me. Besides, as some tools actually share the same command-line structure. This could give you a more convenience.
5.5.2 A batch mode system using decode and decode_anytopo:

What could go wrong?

Let us start with building a batch mode system using `decode` and `decode_anytopo`. They are the first two decoders we will play with. To make your work easy, you need to have a big concept how a recognizer works. There are couple of resource a recognizer need to have before it could run.

- The dictionary In Sphinx 3, that includes a lexical dictionary and a filler dictionary.
- The language model In Sphinx 3, this could be an n-gram as we described or it could also be a finite state machine.
- The acoustic model As we mentioned, there are 5 components of it, the means, the variances, the mixture weight, the transition matrices and the model definition.

In batch mode, you also need to figure out a way to let the recognizer to read in the cepstral files.

Let us think about why the recognizer needs them. Essentially, an HMM-based speech recognizer just tries to compose a graph of states and search the best path within the graph.

Therefore, for the first level, a word graph is required. That could be either an N-gram and FSM.

For the second level, the recognizer needs to know how the word will map to a phone sequence. That’s why a dictionary is necessary. As most large vocabulary recognizer, Sphinx also uses concept of fillers. Major reason is specifying fillers in language model is not feasible and most of the time, special treatment is necessary on fillers. That’s why a separate dictionary is required.

For the final level, the recognizer will need to know how each phone could be represented as state sequences. That’s how the hmm set was used.

The above concept is illustrated in Figure. 5.1.
Figure 5.1: Resource used by the decoders
So, the four options are essentially what you need to have in the decoder. These are the four things you need to first identify. This sounds pretty simple.

“If it is that simple, why it takes me so much time to setup the decoder?” That happens to be many people’s questions. Therefore, it is worthwhile to do some initial check of your resource. In general, you cannot put in arbitrary dictionary, language model, acoustic model and features. Most folks don’t understand that, that’s why most of the time the setup was wrong. So here is a check list you may want to go through first.

**LM/Dictionary Mismatch**

The first situation occurs between inconsistency between language model and dictionary. This is what happens, the language model will try to look up the dictionary and see whether a word exists. If it does, the word will be included in the word graph. If it doesn’t, the word will not be included. This situation could be described as *Language mode-Dictionary mismatch*. All sphinx 3 decoders will warn the users about this. However, it is a design decision not to stop the recognizer being used even the situation exists because it is a very common situation in real life where the dictionary and language model are usually prepared by two different persons.

If you found that happend to you, make sure you add words to the dictionary, then things will be fine.

**Dictionary/AM Mismatch**

Then the next common thing that happens is mismatch between dictionary and the acoustic models. This is what happens. When the phone set used in the dictionary doesn’t match with the phone set used in the acoustic model. The decoder will then stop. This happens a lot when the dictionary are obtained from two sources.

If you see that happen, make sure you do a check of all the phones of dictionary and compare it will what you have in the acoustic model. For models that trained from the recipe of SphinxTrain, this should not happen.

What if you have a model and a dictionary, you know that they have mismatch but you still don’t want to use them together? That happens a lot because most of the time you may not be the one who train the model.
Now what you could do is to map the phones you found in acoustic models or the dictionary such that at the end the two sets are matched. Of course, make sure you know what you are doing. Also, please remind yourself that linguistic knowledge could be wrong in this particular situation. Validation of the model using data should be the final check.

**AM/Feature mismatch**

This problem might be the most subtle and could be the hardest to check. However, this is also the number one killer of most decoding setup occurred in Sphinx Forum. The idea is this unless you know what you are doing, difference between the front-end setting of acoustic model and front end could be detrimental to the recognition result. There are several parameters you will see in `wave2feat` and `livedecode` that could control the front-end. Bear in mind, parameters such as `-cmn`, `-age` and `-varnorm` are also parameters that could control the frontend. If any of them are different, it is not easy to predict what's going on.

In general, when a model is released, to avoid model mismatch in future. It is important for the trainer to also release information about the feature parameters used in training.

**Now, we could run it.**

First convert the waveform to cepstral first, let us called, the input waveform to be `a.wav`.

[ Editor Notes: We need to verify both the arguments of `wave2feat` and `decode` at here ]

```bash
wave2feat
-alpha 0.97
-srate 16000.0
-frate -100
-wlen 0.0256
-nfft 512
-nfilt 40
-lowerf 133.33334
-upperf 6855.49756
-ncep 13
```
The setting follows exactly what one could find in the open source Hub-4 model at

http://www.speech.cs.cmu.edu/sphinx/models/hub4opensrc.jan2002/INFO.ABOUT.MODELS

The setting may not be totally fitted into your situation. First of all, we assume (-mswav yes), that is to say Microsoft wave format. wave2feat could assume input file to be either nist (-nist yes), raw (-raw yes). If you have a batch file, you could also use -c to specify a control file. Usage is trivial.

We also recommend you to specify -dither yes, the reason is that sometimes if the wavefile contains a frame that is all zero. wave2feat could add a small random number to the samples. Without doing so, the frontend could give erroneous result. ¹⁰.

After that, you just need to run decode to run the results.

decode

-lm simple.arpa.DMP

-dict simple.txt

-fdict filler.dict

-mdef hub4opensrc.6000.mdef

-mean means

-var variances

-mixw mixture_weights

-tmat transition_matrices

-ctl simple.control

-cepdir <your cepstral directory>

-agc none

-varnorm no

-cmn current

Inside simple.control, add a into the control file like this

¹⁰Mainly because the log operation.
Make sure that there exists in the directory `your cepstral directory`. The result will then be displayed in the screen. You could also redirect the result to a file. If you think it is too chatty, you can also specify `-hyp <your hypothesis file>` which could give you just the answer. 11.

As we mentioned, you could also use `decode anytopo` instead of `decode`. Starting from 3.6, `decode anytopo` and `decode` have synchronized command-line interface. Therefore, we recommend you to use it instead of the older version. That is to say the usage will be the same for the two executables.

### 5.5.3 A live mode simulator using `livepretend`

Now we make one further step, we now start to simulate a live mode recognition. `livepretend`'s usage is very similar to the use of a combined `wave2feat` AND `decode`. So the usage is very similar to use the tools together. `livepretend` just like `livedecode` only usage of config file but disallow usage of command line arguments. They are the only two applications that have this properties. 12.

There are one interesting thing about using the Sphinx’s live mode recognizer which is it allows the user to backtrack the answer before the sentence is ended. In `livepretend`, you could turn on and off this functionalities by specifying `-phypdump 1/0` in your setting. By default, the code will always give you the partial hypothesis.

What is the use of the partial hypothesis? You may ask. One use of it could be, if you find that a particular keyword is already recognized, you could just stop the recognition without going through.

### 5.5.4 A Live-mode System

So, now we come to the final part of actually using a recognizer. For many of you, it might happen that using the batch mode recognizer already suffices for your purpose. Most people who use Sphinx are either researchers or developers. For researchers, just changing either part of the system and

11. another trick you could use is that you could just grep the pattern FWDVIT from the recognition output

12. This might be something we will want to change it in future. However, backward compatibility will also be maintained if that happened.
test the changes are usually the most important tasks in their lives. In that case, they could either use decode or livepretend. decode is essentially a batch mode decoder. Whereas livepretend could also simulate situation of live-mode decoder. The latter is slightly better for scenario where techniques could possibly make use of all the frames of an utterance.

For developers, using decode with wave2feat, by themselves, already form a very nice software collection to build some types of real-life application. For example, in case your application could afford it. Push-Button application could you to first record the waveform then recognize the waveform. In that case, one could first record the waveform, process it by wave2feat and then pass it by decode. The whole process could just be a replication of the script we mentioned in Section 5.5.2.

For developers though, using the batch mode decoder in an application may not be the best choice all the time. Major reason is delay. Consider this, using the record-then-recognize approach, the response time of the system will equal to the recording time plus the recognition time. If the user speak for 10s, then, let’s say the recognizer also use 10s to recognize (or 1x RT scenario, explained later). Then, it might take 20s for the system to respond. Not many users will be happy about it.

Therefore, for application developers, a more favorable scenario will be while recording some frames in the users speech, start the recognizer to recognize the recorded speech even though it might not be finished. This is usually called live-mode recognition. Most commercial dictation engine works in that way. This is what we want to discus in this subsection.

So how to do it? We would not cover this in detail in this chapter. However, we will give you some ideas and point you to look at further detail in Chapter 12.

Let us just go through the process first. The first step is to do recording and always put it to some kind of buffer of your program. To record the program, you could try to use libs3audio routine, such as cont_ad. If you want to enjoy more flexibility, we would suggest you to use a library called portaudio\(^\text{13}\). The mentioned package could already give the desired result.

The samples should put to a feature extraction routine and the routine will convert the audio samples into cepstral coefficient. The coefficient can then transfer to the recognition routine and recognition will then be performed. Notice that this time only a segment of speech will be decoded one time. What you need to do is just to continue this process until there

\(^{13}\)Not a Sphinx software but it has a pretty liberal licence, you could find the detail in www.portaudio.org
is no speech in the audio buffer.

The above description should give you some idea what you need to do. In Chapter 12, you will also see detail program listing. We will also provide discussion on issues of creating a live-mode applications.

### 5.5.5 Could it be Faster?

Assume you have finished the above exercise of the building a recognition system, it is time to notice something wrong and try to improve it. The first thing you might have notice is that the code might be too slow to be useful. As the end of the whole section, we will provide you a look on how this problem could be tackled.

Search of speech recognition is a very time-consuming process. In Chapter 3, we see that Viterbi algorithm is usually used to efficiently search for the best path in the HMM. In this chapter, we also learned that the actual HMM that will be composed by different level of resource could be huge.

This leaves us a difficult situation. Even with Viterbi algorithm, we need to search in a huge graph with many possible hypotheses. So, it is important know something fundamental on how to speed up the process. In this tutorial, we will explain the concept of beam pruning.

Consider a searching process where multiple paths are being scored in a time-synchronous manner. What one could do in this process is to “prune” or removed paths which have very low score. How could we do it? Here the idea of beam pruning comes to our excuse. We could use the score of the best path as a reference and remove all paths that have scores way lag behind the best path.

Usually, this process is controlled by a parameter called **beam**. If the beam is wider, more paths will be preserved and the search will take longer time. If the beam is narrower, the decoding will usually be faster.

In Sphinx 3 decoder set, including **decode**, **decode anytopo** and **livepretend**. You could use the paramter **-beam** to control the number of paths you want to preserve. By default, it is set to be $1e^{-80}$, If the number becomes larger, the beam width will become smaller. For example, you could consider using $1e^{-60}$ to increase the speed.

Notice that it is a balance to control the size of beam and the accuracy rate you could get. Usually a tuning set is first defined and experiments are performed to determine the best beam-width for a task. Next section will give you a better idea on how this could be done.
Beam pruning is not the only way where the Sphinx 3 decoder set allows user to speed up the recognizer. There are several important features for fast recognition in Sphinx, that includes

1. Fast GMM computation including techniques such frame down-sampling, context-independent model-based GMM selection (CIGMMS) and Gaussian selection

2. Lexical tree implementation that compress the size of the search space by organizing it using a tree-based structure. There are several interesting issues could arise from this structure. Sphinx provides several interesting and practical solution for these issues.

3. Different levels of pruning the beam we mentioned in this chapter is only one type of beam we could use in practice. For example, in a lot of experiments, one find that usage of phone-level and word-level of beam could also be useful as well.

We will advise users to look at Chapter 10 for detail of these techniques.
5.6 Step 6: Evaluate your own system

It is not enough for just building a system. If you would like to know how to make it even better, evaluation of the system is necessary.

One may separate the methods of evaluation of a speech recognition system into two types, on-line and off-line.

In the on-line evaluation, a person will actually call the system and see whether it performs well. Usually, the evaluator will use his subjective judgment to decide whether the system is performing well enough. To make the evaluation more objective, it is reasonable to ask the evaluator to carry out multiple tasks to test different functionalities of the system.

In the off-line evaluation, a collection of recordings from the users will first be obtained. These recordings will then be used to test whether the system performs. There are generally two accepted performance metrics for this type of evaluation. One is sentence error rate and one is word error rate.

It is generally true that using off-line evaluation with large test set will give you a more informative and objective results. However, we could not ignore the usefulness of on-line evaluation in general. This is because some system related issue could only be detected by actually using the system.

We will have some discussions here on which mode of evaluation should be used in your application at this section and generally how to.

5.6.1 On-line Evaluation

There are two distincts issue in testing a real-life speech recognition system. One is to test the speech recognizer's performance and one is to test the application-specific's capability. In our opinion, for most of the time, on-line evaluation could be very useful in the latter case but it might be dangerous in the former case. This is the general reason why. Usually, on-line evaluation rely on subjective judgment of the evaluator on whether the recognizer is good or not. Chances are it could be the acoustic or language models do cover most of the speakers of the target population but the evaluator happens to be an outlier. The evaluator usually has no idea whether this is happening in the evaluation. Therefore, his opinion of the system could easily be biased to factors other than the strength of the system.

Another side of the danger of on-line evaluation is that the evaluator
could be the system programmer. He will then tend to evaluate the system using phrases which are easy to be recognized. This could result in over-optimism about the system.

Despite these problems, on-line evaluation is indispensable to get a “feeling” of the system such as speed and system stability. Though, it is sufficed to say both issue could also be affected by other programs which are running in the background of the applications.

5.6.2 Off-line Evaluation

Off-line evaluation is perhaps a more objective way to carry out evaluation of speech systems. Usually, a set of testing data is first recorded, then the sentence error rate (SER) or word error rate (WER) are then found. Sentence error rate are defined as the percentage of the number of mis-recognized sentence out of all tested utterances. Whereas WER is usually defined as

\[
A = \frac{H - S - I - D}{H} \times 100\%,
\]  

(5.1)

In order to take account of the insertion and deletion errors into computation of accuracy, the word accuracy is defined as,

where

- \( H \) is the total number of words in the reference,
- \( S \) is the number of substitution errors,
- \( I \) is the number of insertions errors and
- \( D \) is the number of deletions errors.

Usually, a dynamic string matching algorithm is used to decide the alignment between the transcription and the decoding output. One program we recommend is sclite developed by NIST


Despite the soundness of the method, it could be possible that a particular collected test set is only biased to certain type of system. Therefore, improvement of system should not be used to optimize on just one test set but on variety of conditions.
5.7 Interlude

Now you have build your first system, you sweat to collect all resources and incorporate them with the speech recognizer. You also evaluated it and get a number as a performance metric. This is usually not the most satisfactory result. As we explained, the model you used could be trained in a totally different conditions and might not be useful in your situation. For example, one common problem is that what one could get is only a 16K models but the actual scenario requires a 8k setup. What you could do is probably to recalculate the position filter banks. However, it doesn't always give you the best result you could have.

This is the time you may want to train a new model. However, it is important to realize that training could be harder than what you think. There are several processes that could take you a long time. These are unfortunately not fully understood by many novices of speech recognition system. 14

In general, the actually training procedure, compared to the preparation process such as data collection and transcription, are usually the least laborious. Therefore, instead of giving you any real technical advice first, we will recommend you to be psychologically prepared before the process. Because no matter how detail and careful our instruction are, the process of training still takes quite an amount of time.

In the rest of this chapter, we will go through the basic process of training. We will not cover everything and will point you to some later chapters for further details. We will consider alternatives of some laborous process and you could decide which way to go.

5.8 Step 7: Collection and Processing of Data

From now, we start to train a model from scratch from collecting data to create the final acoustic models. So, let us try to see the process from the beginning. That is how data is collected in the first place in speech recognition.

Essentially, what we need to do is just to collect some data and record it as waveforms. The tricky part is that

- It is better to collect as much data as possible.

14Worst of all, when they have no one to blame. They blamed the recognizer.
• The waveforms have to be recorded which are similar to the condition in the actual decoding.

5.8.1 Determine the amount of data needed

How much recordings are needed? Most of the time, at least 1 hour of speech is required to a basic model. If a model is supposed to be used for speaker-independent speech recognition for a large population, perhaps more hours of speech is required.

The amount of data needs to be collected also relates to the parameter of the acoustic model.

Here is a real life example, if every Gaussian distribution requires 100 data points to train. With a system with 1000 senone, each senone is represented by 8 Gaussian, then loosely speaking \(^{15}\) In that case 800000 frames of speech will be required. In that case, assuming that 1000 frames mean 10s of speech (This depends on the sampling rate and frame rate). You will need have 800 utterance which solely composed of speech. Now, let’s say we assume that the speech density is 50%. Then 1600 utterances of waveforms will be needed. That if you do the calculation that will approximately \(\frac{1600 \times 10}{60+60} \approx 6\) hours of speech.

It is also important to make sure that the recordings are phonetically balanced. What it means is it is better to make sure that the amount of training data that will train each phone should be balanced. For example, let’s follow the above examples and you did collect 1600 utterances but all of them are “Hello”, then in this case, only 4 phones will be trained. (The number of triphones are also pretty limited). Then, other phones in your phone list will be totally empty in your acoustic models. That usually will cause the training process behave abnormally one way or the other.

5.8.2 Record the data in appropriate condition: environmental noise

To explain what it means by appropriate, lets consider the following very simple example, model trained by data collected in an office room would likely perform badly in a condition where there are construction noise

\(^{15}\)it’s loose because Baum-Welch algorithm will give every states a fractional counts for training.
around. (And so does a human audio system.) Vice versa. In general, the environment usually change the speech characteristic.

In a signal processing term, there are usually two different types of noise. One is additive noise and the other is convolutive noise.

Additive noises are usually caused by external sound sources such as construction noise, babble noise, gun noise. The sound wave superimpose the speech and causing the receiver machine record some speech which is different from the original sound source.

Convolutive noise is also called channel noise. For example, when a sound wave pass through a media such as air. Or recorded through a microphone. Their characteristic will usually change simply because different frequency component of the sound wave response differently to the channel. Or in another words, the channel forms a filter to the signal.

From the above discussion, it is easy for us to understand why most dictation engine’s instruction will usually instruct the users to record the speech in a quiet environment. Some vendor will even provide users a prepared microphones. Likely, this is not really because the microphone is better than the other but because the type of microphone has been used consistently in collection process.

For developers, this implies that data collection could be the first challenge one needs to deal with. For a desktop application, a quiet room and consistently using same type of microphones would suffice. It is also important to instruct the users to speak to the microphone with constant distance especially headset microphone.

\[16\] The algorithm could actually account for additive noise
5.9 Step 8: Transcription of data
5.10 Step (7 & 8)b : If you don’t want to sweat
5.11 Step (7 & 8)c : If you don’t want to sweat and you have no money
5.12 Step 9: Training of acoustic model. Follow a recipe.
5.13 Step 10: Training of acoustic model. Do it from scratch.
5.14 Step 11: Trace the source code of Sphinx

[Editor Notes: We should also add a chapter of LM training]
5.15 How do I get more help?
Part III

CMU Sphinx Speech Recognition System
Chapter 6

Sphinx’s Front End

Author: Mike Seltzer, Evandro Gouvêa and Arthur Chan, Editor: Arthur Chan

[Editor Notes: This is started from Mike’s Sphinx III Signal Processing Front End Specification]

6.1 Introduction

This chapter describes the signal processing front end of the Sphinx III Speech recognition system. The front end transforms a speech waveforms into a set of features to be used for recognition. The current implementation creates mel-frequency cepstral coefficient (MFCC).
6.2 Block Diagram

Below is a block diagram of feature extraction operations performed by the Sphinx III front end.

Speech Waveform (16 bits) → Front End Processing Parameters

- Pre-emphasis
- Framing
- Windowing
- Power Spectrum
- Mel Spectrum
- Mel Cepstrum

Frame Based Processing

Mel Frequency Cepstral Coefficients (32 bit floating point)
6.3 Front End Processing Parameters

The following parameter structure must be completed by the user prior to using the front end. Any parameter that is set to 0 will be set to its default value.

[Editor Notes: The structure is not displayed correctly yet {}]

[EBG: why are you presenting a struct here (internal to the code)? Why not something useful to the end user?]

typedef struct
  float32 SAMPLING_RATE
  int32 FRAME_RATE
  float32 WINDOW_LENGTH
  int32 FB_TYPE
  int32 NUM_CEPSTRA
  int32 NUM_FILTERS
  int32 FFT_SIZE
  float32 LOWER_FILT_FREQ
  float32 UPPER_FILT_FREQ
  float32 PRE_EMPHASIS_ALPHA
} param_t;

6.4 Detail of Front End processing

6.4.1 Pre-emphasis

The following finite impulse response pre-emphasis filter is applied to input waveforms:

\[ y[n] = x[n] - \alpha x[n-1] \]

\( \alpha \) is provided by the user or set to the default value. If \( \alpha = 0 \), then this step is skipped. In addition, the appropriate sample of the input is sorted as a history value for use during the next round of processing.
6.4.2 Framing

Framing will depend on the windows length (in terms of second) and the frame rate (in terms of frames/second).

[ Editor Notes: Under construction]

6.4.3 Windowing

The frame is multiplied by the following Hamming window:

\[ w[n] = 0.54 - 0.46 \cos \left( \frac{2\pi n}{N-1} \right) \]

\( N \) is the length of the frame.

6.4.4 Power Spectrum

The power spectrum of the frame is computed by performing a discrete Fourier Transform of length specified by the user, and then computing its magnitude squared.

\[ S[k] = (Re(X[k]))^2 + (Im(X[k]))^2 \]

6.4.5 Mel Spectrum

The mel spectrum of the power spectrum is computed by multiplying the power spectrum by each of the triangular mel weighting filters and integrating the results:

\[ \tilde{S}[l] = \sum_{k=0}^{N/2} S[k] M_l[k] \quad l=0,1,\ldots,L-1 \]

where \( N \) is the length of the DFT and \( L \) is the total number of triangular mel weighting filters.

6.4.6 Mel Cepstrum

A DCT is applied to the natural logarithm of the mel spectrum to obtain the mel cepstrum:

\[ c[n] = \sum_{i=0}^{L-1} \ln(\tilde{S}[i]) \cos\left( \frac{\pi n}{2L}(2i + 1) \right) \quad c=0,1,\ldots,C-1 \]

\( C \) is the number of cepstral coefficients.


6.5 Mel Filter Banks

The mel scale filterbank is a series of \( L \) triangular bandpass filters that have been designed to simulate the bandpass filtering believed to occur in the auditory system. This corresponds to a series of bandpass filters with constant bandwidth and spacing on a mel frequency scale. On a linear frequency scale, this filter spacing is approximately linear up to 1kHz and logarithmic at higher frequencies. The following warping function transforms linear frequencies to mel frequencies:

\[
mel(f) = 2595 \log(1 + \frac{f}{700})
\]

A plot of the warping function is shown below.

[ Editor Notes: Should include Mike's diagram here]

A series of \( L \) triangular filters with 50\% overlap are constructed such that they are equally spaced on the mel scale spanning \([\mel(f_{\text{min}}), \mel(f_{\text{max}})]\) where \( f_{\text{min}} \) and \( f_{\text{max}} \) can be set by the user. The triangles are all normalized so that they have unit area.

6.6 The Default Front End Parameters

These are the default values for the current Sphinx III front end:

<table>
<thead>
<tr>
<th>parameter</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Rate</td>
<td>16000.0Hz</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>100 Frames/sec</td>
</tr>
<tr>
<td>Window Length</td>
<td>0.025625 sec</td>
</tr>
<tr>
<td>Filterbank</td>
<td>Mel Filterbank</td>
</tr>
<tr>
<td>Number of Cepstra</td>
<td>13</td>
</tr>
<tr>
<td>Number of Mel Filters</td>
<td>40</td>
</tr>
<tr>
<td>DFT Size</td>
<td>512</td>
</tr>
<tr>
<td>Lower Filer Frequency</td>
<td>133.33334 Hz</td>
</tr>
<tr>
<td>Upper Filer Frequency</td>
<td>6855.4976 Hz</td>
</tr>
<tr>
<td>Pre-Emphasis ( \alpha )</td>
<td>0.97</td>
</tr>
</tbody>
</table>
6.7 Computation of Dynamic Coefficients

Author: Arthur Chan, Editor: Arthur Chan

There are three major types of mechanism for generation of dynamic coefficients. For space efficiency, the dynamic coefficients computation are usually carried out at the routine which directly uses the feature, for example, the decoders `decode` or `decode_anytopo`.

There are two sets of notation to represent the mechanism of dynamic coefficient generation, one for the decoder and one for the trainer. There is actually a one to one mapping between the two notations. The following table will be indispensable for one to understand the two notations.

The dynamic coefficient types s2_4x, s3_1x39, and 1s_c_d_dd are commonly used in Sphinx III. Currently other coefficient types, 1s_c, 1s_c_d, 1s_c_dd, and 1s_c_d_dd, [EBG: HUH???] are obsolete. They are currently safeguarded in various recognizers.

<table>
<thead>
<tr>
<th>Sphinx 2</th>
<th>s2_4x</th>
</tr>
</thead>
<tbody>
<tr>
<td>SphinxTrain, simplified</td>
<td>4s_12c_24d_3p_12dd</td>
</tr>
<tr>
<td>SphinxTrain, generic</td>
<td>c/1..L-1/d/1..L-1/c/0/d/0/dd/0/,dd/1..L-1/</td>
</tr>
<tr>
<td>vector length</td>
<td>4 in parallel, dimensions 12, 24, 3, and 12</td>
</tr>
<tr>
<td>feature layout</td>
<td>12 ceps, 12 short term Δ ceps, 12 long term Δ ceps, pow, short term Δ pow, ΔΔ pow, 12 ΔΔ ceps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sphinx 3</th>
<th>s3_1x39</th>
</tr>
</thead>
<tbody>
<tr>
<td>SphinxTrain, simplified</td>
<td>1s_12c_12d_3p_12dd</td>
</tr>
<tr>
<td>SphinxTrain, generic</td>
<td>c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/</td>
</tr>
<tr>
<td>vector length</td>
<td>39</td>
</tr>
<tr>
<td>feature layout</td>
<td>12 ceps, 12 Δ ceps, pow, Δ pow, ΔΔ pow, 12 ΔΔ ceps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sphinx 3</th>
<th>1s_c_d_dd</th>
</tr>
</thead>
<tbody>
<tr>
<td>SphinxTrain, simplified</td>
<td>1s_c_d_dd</td>
</tr>
<tr>
<td>SphinxTrain, generic</td>
<td>c/0..L-1/d/0..L-1/dd/0..L-1/</td>
</tr>
<tr>
<td>vector length</td>
<td>39</td>
</tr>
<tr>
<td>feature layout</td>
<td>13 ceps (i.e. power + 12 cepstrae), 13 Δ ceps, 13 ΔΔ ceps</td>
</tr>
</tbody>
</table>
6.7.1 Generic and simplified notations in SphinxTrain

Because of code legacy, SphinxTrain actually maintained two sets of notations that are equivalent but written differently. The following table will allow one to inter-convert them.

<table>
<thead>
<tr>
<th>Simplified</th>
<th>Generic</th>
<th>sphinx3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c/1..L-1/.d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/</td>
<td>4s_12c_12d_3p_12dd</td>
<td>s2_4x</td>
</tr>
<tr>
<td>c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/</td>
<td>1s_12c_12d_3p_12dd</td>
<td>s3_1x39</td>
</tr>
<tr>
<td>c/0..L-1/d/0..L-1/dd/0..L-1/</td>
<td>1s_c_d_dd</td>
<td>1s_c_d_dd</td>
</tr>
<tr>
<td>c/0..L-1/d/0..L-1/</td>
<td>1s_c_d</td>
<td>-</td>
</tr>
<tr>
<td>c/0..L-1/</td>
<td>1s_c</td>
<td>-</td>
</tr>
<tr>
<td>c/0..L-1/dd/0..L-1/</td>
<td>1s_c_d</td>
<td>-</td>
</tr>
</tbody>
</table>

6.7.2 Formulae for Dynamic Coefficient

Let \( x[i] \) be the incoming cepstra at time \( i \), \( c[i] \), the cepstral vector, \( d[i] \), the delta cepstral vector, \( i.e. \) the first derivative, and \( dd[i] \), the delta delta cepstral vector, \( i.e. \) the second derivative, all at time \( i \). The dynamic coefficients for \( s3_1x39 \) and \( 1s_c_d_dd \) are computed as:

\[
\begin{align*}
  c[i] &= x[i] \\
  d[i] &= x[i + 2] - x[i - 2] \\
  dd[i] &= d[i + 1] - d[i - 1] = (x[i + 3] - x[i - 1]) - (x[i + 1] - x[i - 3])
\end{align*}
\]

6.7.3 Handling of Initial and Final Position of an Utterance

The above formulae will require that, for every frame, the previous 3 frames and the future 3 frames exist. So, special treatment is required to pad 3 frames at the beginning of an utterance and 3 frames at the end of an utterance. Sphinx 3.x takes care of it by replicating the first three frames of an utterance once. The sequence of features \( i.e. \) \( c[i], d[i], dd[i] \) will be delayed relative to the incoming cepstra. Moreover, the first couple of derivative coefficients are computed as if the sequence of frames had been converted from:

\[
\]

to:

\[
\]
Therefore, when actually viewing the frames (say dumping the code by printf), the first frame will have the delta and delta delta coefficients to be zero, whereas the second frame will have the delta coefficients to be zero.

### 6.8 Cautions in Using the Front End

Several common pitfalls of using the front end.

- The user is advised to use dither (**-dither**) for waveform decoding. In Sphinx 3.X's wave2feat, this process is completely repeatable and it is controlled by the option **-seed**. The dither is useful because frames filled with zeros could cause the front end to run into a divide by zero condition.

- The user is strongly advised to match the front end parameters in both decoding and training. \(^1\)

### 6.9 Compatibility of Front Ends in Different Sphinxen

The cepstral generation source code is shared by Sphinx II, III and Sphinx-Train. However, the dynamic coefficients routine in Sphinx III only supports types s3_1x39 and 1s_c_d_dd. Moreover, Sphinx II is hardwired to use the type s2_4x. Therefore, if you are training models for Sphinx II, you will have to use this same type in SphinxTrain.

### 6.10 Cepview

One useful tool for viewing the cepstral coefficient is **cepview**. It can be found in sphinx 3.X (X > 5).

For example, the following command can be used to view the feature file **a.mfcc**.

\(^1\)There are workarounds if the sampling frequency used in the training is higher than in the decoding. However, the upper and lower frequency in the filter bank need to scale for the decoding's frequency in that case. Avoid this unless you know what you are doing.
$ cepview -d 13 -describe 1 -header 1 -f a.mfcc

You will see

$ 329 frames


  0: 5.916 -0.248 0.128 0.083 -0.143 0.254 -0.012 -0.327 -0.20 0 -0.004 0.020 -0.209 -0.037
  1: 5.668 -0.203 0.235 0.123 -0.184 0.145 0.031 -0.053 -0.224 -0.050 -0.087 -0.252 -0.086
  2: 5.371 -0.159 0.271 0.001 -0.055 0.242 -0.011 -0.071 -0.144 0.151 0.096 -0.187 -0.175
  3: 5.499 -0.182 0.301 0.071 0.093 -0.036 -0.263 -0.164 -0.295 0.132 -0.079 -0.070 0.018
  4: 5.447 -0.307 0.241 -0.067 -0.071 0.177 -0.050 -0.070 -0.327 0.163 0.046 0.084 -0.070
  5: 5.515 -0.070 0.062 -0.113 -0.008 0.180 -0.079 -0.073 -0.282 0.006 -0.082 -0.028 -0.123
  6: 5.431 -0.107 -0.007 -0.133 -0.037 0.137 -0.145 -0.089 -0.144 0.069 -0.085 0.016 -0.165
  7: 5.502 -0.094 0.162 -0.142 -0.047 0.077 -0.073 -0.086 -0.246 -0.074 -0.182 0.050 0.052
Chapter 7

General Software Description of SphinxTrain

Author: Arthur Chan, Editor: Arthur Chan

The acoustic model training package, SphinxTrain, is a powerful set of routines that could be used to train several types of hidden Markov model for speech recognition.

As as Aug 2005, SphinxTrain consists of 35 C-based applications and 9 perl scripts. For many practitioners, its enormous size was the core difficulty to learn it. This chapter provides an overview of them and it is recommended for the users to read this chapter before reading Chapter 8: “Acoustic Model Training”.

7.1 On-line Help of SphinxTrain

Tools of SphinxTrain have coherent design. They behave the same in the application level and there are common commands which would help the users to use them.

First of all, if the user just invoke the command without inputting any command-line arguments. Then a list of help for the command-line options will be displayed.

For example, for the tool printp, we would have

$ printp
[Switch] [Default] [Description]
-help no Shows the usage of the tool
-example no Shows example of how to use the tool
-tmatfn The transition matrix parameter file name
-mixwfn The mixture weight parameter file name
-mixws Start id of mixing weight subinterval
-mixwe End id of mixing weight subinterval
-gaufn A Gaussian parameter file name
-gaucntfn A Gaussian parameter weighted vector file
-regmatcntfn MLLR regression matrix count file
-moddeffn The model definition file
-lambdafn The interpolation weight file
-lambdamin 0 Print int. wt. >= this
-lambdamax 1 Print int. wt. <= this
-norm yes Print normalized parameters
-sigfig 4 Number of significant digits in 'e' notation

If the user needs further help, each routine provides two important mechanism for the user. One is the option -help and one is the option -example. The first one give a written description of each tool. For example,

printp -help yes

will produce the current description of the tool

./bin.i686-pc-linux-gnu/printp
-help yes

Description:
Display numerical values of resources generated by Sphinx
Current we support the following formats
-tmatfn : transition matrix
-mixwfn : mixture weight file
-gaufn : mean or variance
-gaucntfn : sufficient statistics for mean and diagonal covarian
-lambdafn : interpolation weight
Currently, some parameters can be specified as intervals such as mixture
You can also specified -sigfig the number of significant digits by you
and normalize the parameters by -norm

whereas

$ printp -example yes

will produce

Example:
Print the mean of a Gaussian:
printp -gaufn mean
Print the variance of a Gaussian:
printp -gaufn var
Print the sufficient statistic:
printp -gaucntfn gaucnt:
Print the mixture weights:
printp -mixw mixw
Print the interpolation weight:
printp -lambdafn lambda

The help and example message of SphinxTrain’s tools are also generated automatically and converted into instruction manual which one could find in the appendix A.2 of this manual.

7.2 Basic Acoustic Models Format in Sphinx-Train

In all Sphinxen (Sphinx II, Sphinx III and Sphinx IV), the basic acoustic model unit is the HMM set. However, for efficiency purpose, the HMM set is usually decomposed into several components as in representation. This is quite different from other software package such as HTK and ISIP recognizer.

For example, in Sphinx III, the models are represented by a set of means, variances, mixture weights, transition matrices and the model.
definition. These method of decomposition allows researchers to just ma-
ipulate one type of quantity instead of all files.

You could find more information of the acoustic model format in the next Section

7.3 Sphinx data and model formats

7.3.1 Sphinx II data formats

Note: The data format described here is obsolete

# Feature set: This is a binary file with all the elements in each of the vectors stored sequentially. The header is a 4 byte integer which tells us how many floating point numbers there are in the file. This is followed by the actual cepstral values (usually 13 cepstral values per frame, with 10ms skip between adjacent frames. Framesize is usually fixed and is usually 25ms).

<4 byte integer header>
vec 1 element 1
vec 1 element 2
.
.
vec 1 element 13
vec 2 element 1
vec 2 element 2
.
.
vec 2 element 13

7.3.2 Sphinx II Model formats

Author: Rita Singh and Arthur Chan, Editor: Arthur Chan

Sphinx II semi-continuous HMM (SCHMM) formats:
The Sphinx II SCHMM format is rather complicated. It has the following main components (each of which has sub-components):

1. A set of codebooks
2. A "sendump" file that stores state (senone) distributions
3. A "phone" and a "map" file which map senones on to states of a triphone
4. A set of ".chmm" files that store transition matrices

Here is their description

**Codebooks**

There are 8 codebook files. The sphinx-2 uses a four stream feature set:

- cepstral feature: \([c1-c12]\), (12 components)
- delta feature: \([\text{delta}_c1-\text{delta}_c12, \text{longterm}_{\text{delta}}_c1-\text{longterm}_{\text{delta}}_c12]\), (24 components)
- power feature: \([c0, \text{delta}_c0, \text{doubledelta}_c0]\), (3 components)
- doubledelta feature: \([\text{doubledelta}_c-\text{doubledelta}_c12]\) (12 components)

The 8 codebooks files store the means and variances of all the gaussians for each of these 4 features. The 8 codebooks are,

- cep.256.vec [this is the file of means for the cepstral feature]
- cep.256.var [this is the file of variances for the cepstral feature]
- d2cep.256.vec [this is the file of means for the delta cepstral feature]
- d2cep.256.var [this is the file of variances for the delta cepstral feature]
- p3cep.256.vec [this is the file of means for the power feature]
- p3cep.256.var [this is the file of variances for the power feature]
- xcep.256.vec [this is the file of means for the double delta feature]
- xcep.256.var [this is the file of variances for the double delta feature]
All files are binary and have the following format: [4 byte int][4 byte float][4 byte float][4 byte float]...... The 4 byte integer header stores the number of floating point values to follow in the file. For the cep.256.var, cep.256.vec, xcep.256.var and xcep.256.vec this value should be 3328. For d2cep.* it should be 6400, and for p3cep.* it should be 768. The floating point numbers are the components of the mean vectors (or variance vectors) laid end to end. So cep.256.[vec,var] have 256 mean (or variance) vectors, each 13 dimensions long, d2cep.256.[vec,var] have 256 mean/var vectors, each 25 dimensions long, p3cep.256.[vec,var] have 256 vectors, each of dimension 3, xcep.256.[vec,var] have 256 vectors of length 13 each.

The 0th component of the cep,d2cep and xcep distributions are not used in likelihood computation and are part of the format for purely historical reasons.

The “sendump” file

The “sendump” file stores the mixture weights of the states associated with each phone. (this file has a little ascii header, which might help you a little). Except for the header, this is a binary file. The mixture weights have all been transformed to 8 bit integer by the following operation intmixw = (-log(float mixw) ¿¿ shift) The log base is 1.0003. The ”shift” is the number of bits the smallest mixture weight has to be shifted right to fit in 8 bits. The sendump file stores,

for each feature (4 features in all)
for each codeword (256 in all)
for each ci-phone (including noise phones)
for each tied state associated with ci phone,
probability of codeword in tied state
end
for each CI state associated with ci phone, ( 5 states )
probability of codeword in CI state
end
end
end
The sendump file has the following storage format (all data, except for the header string are binary):

Length of header as 4 byte int (including terminating ‘0’)

HEADER string (including terminating ‘0’)
0 (as 4 byte int, indicates end of header strings).
256 (codebooksize, 4 byte int)
Num senones (Total number of tied states, 4 byte int)
[lut[0], (4 byte integer, lut[i] = -(i"<<"shift))
prob_of_codeword[0].of_feat[0].1st_CD_sen.of.1st_ciphone (uchar)
prob_of_codeword[0].of_feat[0].2nd_CD_sen.of.1st_ciphone (uchar)
...
prob_of_codeword[0].of_feat[0].1st_CI_sen.of.1st_ciphone (uchar)
prob_of_codeword[0].of_feat[0].2nd_CI_sen.of.1st_ciphone (uchar)
...
prob_of_codeword[0].of_feat[0].1st_CD_sen.of.2nd_ciphone (uchar)
prob_of_codeword[0].of_feat[0].2nd_CD_sen.of.2nd_ciphone (uchar)
...
prob_of_codeword[0].of_feat[0].1st_CI_sen.of.2nd_ciphone (uchar)
prob_of_codeword[0].of_feat[0].2nd_CI_sen.of.2nd_ciphone (uchar)
...
]
[lut[1], (4 byte integer)
prob_of_codeword[1].of_feat[0].1st_CD_sen.of.1st_ciphone (uchar)
prob_of_codeword[1].of_feat[0].2nd_CD_sen.of.1st_ciphone (uchar)
..
prob_of_codeword[1].of_feat[0].1st_CD_sen.of.2nd_ciphone (uchar)
prob_of_codeword[1].of_feat[0].2nd_CD_sen.of.2nd_ciphone (uchar)
..
]
... 256 times ..

Above repeats for each of the 4 features
**PHONE file**

The phone file stores a list of phones and triphones used by the decoder. This is an ascii file. It has 2 sections. The first section lists the CI phones in the models and consists of lines of the format. For example:

```
AA 0 0 8 8
```

"AA" is the CI phone, the first "0" indicates that it is a CI phone, the first 8 is the index of the CI phone, and the last 8 is the line number in the file. The second 0 is there for historical reasons.

The second section lists TRIPHONES and consists of lines of the format:

```
A(B,C)P -1 0 num num2
```

"A" stands for the central phone, "B" for the left context, and "C" for the right context phone. The "P" stands for the position of the triphone and can take 4 values "s", "b", "i", and "e", standing for single word, word beginning, word internal, and word ending triphone. The -1 indicates that it is a triphone and not a CI phone. num is the index of the CI phone "A", and num2 is the position of the triphone (or ciphone) in the list, essentially the number of the line in the file (beginning with 0).

**The "map" file**

The "map" file stores a mapping table to show which senones each state of each triphone are mapped to. This is also an ascii file with lines of the form:

```
AA (AA, AA) s<0> 4
AA (AA, AA) s<1> 27
AA (AA, AA) s<2> 69
AA (AA, AA) s<3> 78
AA (AA, AA) s<4> 100
```

The first line indicates that the 0th state of the triphone "AA" in the context of "AA" and "AA" is modelled by the 4th senone associated with the CI phone AA. Note that the numbering is specific to the CI phone. So the 4th senone of "AX" would also be numbered 4 (but this should not cause confusion).
There is one *.chmm file per ci phone. Each stores the transition matrix associated with that particular ci phone in following binary format. (Note all triphones associated with a ci phone share its transition matrix) (all numbers are 4 byte integers):

- -10 (a header to indicate this is a tmat file)
- 256 (no of codewords)
- 5 (no of emitting states)
- 6 (total no. of states, including non-emitting state)
- 1 (no. of initial states. In fbs8 a state sequence can only begin with state[0]. So there is only 1 possible initial state)
- 0 (list of initial states. Here there is only one, namely state 0)
- 1 (no. of terminal states. There is only one non-emitting terminal state)
- 5 (id of terminal state. This is 5 for a 5 state HMM)
- 14 (total no. of non-zero transitions allowed by topology)

And also

```
[0 0 (int)log(tmat[0][0]) 0] (source, dest, transition prob, source id)
[0 1 (int)log(tmat[0][1]) 0]
[1 1 (int)log(tmat[1][1]) 1]
[1 2 (int)log(tmat[1][2]) 1]
[2 2 (int)log(tmat[2][2]) 2]
[2 3 (int)log(tmat[2][3]) 2]
[3 3 (int)log(tmat[3][3]) 3]
[3 4 (int)log(tmat[3][4]) 3]
[4 4 (int)log(tmat[4][4]) 4]
[4 5 (int)log(tmat[4][5]) 4]
[0 2 (int)log(tmat[0][2]) 0]
[1 3 (int)log(tmat[1][3]) 1]
[2 4 (int)log(tmat[2][4]) 2]
```
There are thus 65 integers in all, and so each *.chmm file should be 65*4 = 260 bytes in size.

7.3.3 Sphinx III model formats

Sphinx III models are composed of 5 parts

- Model Definition
- Means
- Variances
- Mixture Weights
- Transition Matrices

Headers of all binary formats

Model Definition

[Editor Notes: Plainly copied from the recipe chapter, need to rewrite] Here is an example:

0.3
55 n_base
118004 n_tri
472236 n_state_map
2165 n_tied_state
165 n_tied_ci_state
55 n_tied_tmat
#
# Columns definitions
#base lft rt p attrib tmat ... state id’s ...
+BACKGROUND+ - - - filler 0 0 1 2 N
+BREATH+ - - - filler 1 3 4 5 N
+CLICKS+ - - - filler 2 6 7 8 N
The first few line describes the number of parameters in the whole system

0.3
0.3 is the model format version.

55 n_base

\( n_{\text{base}} \) is the number of base phone including fillers in the system. In this case, the number is 55.

118004 n_tri

\( n_{\text{tri}} \) is the number of triphone in the system. In this case, the number is 118004.

472236 n_state_map

2165 n_tied_state

\( n_{\text{tied\_state}} \) is the number of triphone in the system. In this case, the number is 118004.

165 n_tied_ci_state

\( n_{\text{tied\_ci\_state}} \) is the number tied CI state in the system. In this case, the number is 165

55 n_tied_tmat

\( n_{\text{tied\_tmat}} \) is the number tied transition matrix in the system. In this case, the number is 55.

Now let us try to interpret the following line.

#base lft rt p attrib tmat ... state id’s ...

+BACKGROUND+ - - - filler 0 0 1 2 N

From left to right, it reads the base phone is +BACKGROUND+, no left context and no right context, so it is a CI phone. It is a filler. Its phone ID is 0, its first state has senone ID 0, second state 1, third state 2.

It is quite useful to know how the hmm state would map into a senone. (In HTK terminology, the tied-state.) In the above example, if you know that the senone ID for first state is 0. You just need to look up the file means and variances to get the value of them.

AA - - - n/a 11 33 34 35 N

Here is another entry in the model definition file. This time, we see a typical CI phone entry. If you used the standard training procedure of SphinxTrain (that you will also have a chance to play with in the second half of this chapter), then all the CI phone HMM’s state will have their own senone which is unique for it.

As opposed to that, if you look at a triphone definition.

AA AA AA s n/a 11 177 190 216 N
You will see multiple phones states are actually mapped to the same senone ID. Why does that happen? This is essentially the result of tying happened in training the context-dependent model. What happen is there are always not enough training data for triphone model. Therefore, it is necessary to cluster the hmm-state to make data could be more efficiently used. Most of the time, clustering are either done by decision tree-base clustering or bottom-up agglomerative clustering.

**Means, Variances, MixtureWeights and Transition Matrices**

[Editor Notes: Need to fill in more information here.]

Please read the chapter on recipe for further detail.

### 7.3.4 Sphinx IV model formats

To create a Sphinx IV model, a common way is generate a Sphinx III models through SphinxTrain and convert it to Sphinx IV format. One could find detail information at

7.4 Software Architecture of SphinxTrain

Author: Arthur Chan, Editor: Arthur Chan

Based on them, a 9-step procedure usually called “SphinxTrain perl script” was built. If one look at them, it looks like

7.4.1 General Description of the C-based Applications

As at Aug 2005, there are 35 tools in SphinxTrain, they are

agg_seg init_mixw mk_model_def mk_s3tmat
bldtree kmeans_init mk_s2cb mk_ts2cb
bw makequests mk_s2hmm mllr_solve tiestate
cp_parm map_adapt mk_s2phone mllr_transform wave2feat
delint mixw_interp mk_s2phonemap norm
dict2tri mk_flat mk_s2sendump param_cnt
inc_comp mk_mdef_gen mk_s3gau printp
init_gau mk_mllr_class mk_s3mixw prunetree

It is perhaps very hard to describe them one by one without a proper categorization. Here is one categorization that was widely used.

1. Model conversion routines
   That includes \texttt{mk\_s2cb}, \texttt{mk\_s2hmm}, \texttt{mk\_s2phone}, \texttt{mk\_s2phonemap}, \texttt{mk\_s2sendump}, they are for Sphinx 3 models format to Sphinx 2 model format.
   Also \texttt{mk\_s3gau}, \texttt{mk\_s3mixw}, \texttt{mk\_s3tmat}, \texttt{mk\_ts2cb}, they are for conversion of Sphinx 2 model format to Sphinx 3 model format.

2. Model manipulation routines
   That includes \texttt{printp}, which could display acoustic model parameters as well some important data structure.
   \texttt{cp\_parm} which could copy model parameters.
   \texttt{inc\_comp} which could increase the number of mixture components.
   \texttt{mk\_model\_def}, \texttt{mk\_mdef\_gen} which could create the model definition file.

\footnote{The tool \texttt{QUICK\_COUNT} also exists, but it is largely replaced by dict2tri}
3. Model adaptation routines
   - `mk_mllr_class` which could create regression classes.
   - `mllr_transform` which could transform a set of model based on a set of regression classes.
   - `map_adapt` which could adapt a set of model based on the maximum a posterior (MAP) criterion.
   - `mllr_solve` which could compute the regression matric based on sufficient statistics of the adaptation data.

4. Feature extraction routine `wave2feat` which could transform a wave file to cepstral coefficient.

5. Desicion tree mainipulation routine
   - `bldtree` which could create a decision tree based on a set of questions.
   - `tiestate` which could tie the state based on the decision trees.
   - `prunetree` which could prune a decision tree to make it has a more reasonable size.
   - `makequests` which could automatically generate a set of questions.

6. Model Initialization routines `mk_flat` would could be used for flat initialization `init_gau, init_mixw` which could be used for ?? `agg_seg, kmeans_init` which could be used for mainipulation.

7. Model Estimation routines
   - `bw` which could be used to carry out Baum-Welch estimation and generate the sufficient statistics.
   - `norm` which could collect the sufficient statistics and generate the model.
   - `delint` which could carry out deleted interpolation.
   - `mixw_interp` which could carry out mixture weight interpolation

8. Miscellaneous routines `dict2tri` which could convert the transcription to triphone.
   - `param_cnt` which could count the parameters out of several resources.

### 7.4.2 General Description of the Perl-based Tool

00.verify 06.prunetree
01.vector_quantize 07.cd-schmm
The current script in SphinxTrain could carry out both fully-continuous HMM training and semi-continuous training. The user is advised to first follow a recipe of training before they start from scratch. One well-maintained tutorial could be found at

http://www.cs.cmu.edu/robust/Tutorial/opensource.html
Chapter 8

Acoustic Model Training

Author: *Rita Singh, Evandro Gouvea and Arthur Chan*, Editor: *Arthur Chan*

[Editor Notes: This chapter contains both Rita’s original tutorial (largely maintained by Evandro).]

[Editor Notes: Also contains and interesting article written by me that talks about that]

[Editor Notes: some common problem factors could cause problems in the training process. Possibly, we should move it to Appendix.]

[Editor Notes: In general, we should put more example instead of command line description in this chapter. It sounds like too much]

Acoustic model training is perhaps the hardest part in the process of building a speech recognition system. For many novices, the convenience provided by the implementation Sphinx sometimes confuse them. Therefore, this chapter tries to provide some in-depth description of the concept and the procedure of acoustic model training and hopefully could give the user an easier time.

Despite the comprehensive nature of this chapter, the user is recommended to first read Chapter 7 to get a feeling of the software SphinxTrain first. This will give them a more concrete feeling of how things are done in the framework.

Here is a brief description of the organization of this chapter, we will first describes several important concepts in training, such as continuous HMM and sem-continous HMM, Baum-Welch algorithm, flat initialization, decision tree tying and mixture growing Section 8.1. These fundamental concepts will be crucial in understanding how training should be done. This will enable the user to have a feeling of how the tools work in general. The rest of the chapter will describe how the procedure should be proceeded. In Section 8.2, preparation of training will be described. In Section 8.3, training procedure of fully continuous HMM will be described.
8.1 Basic Concepts of Training

8.1.1 Flat Initialization

8.1.2 Continuous and Semi-continuous HMM

8.1.3 Baum-Welch algorithm

8.1.4 Decision Tree tying

8.1.5 Mixture Growing

   Author: Arthur Chan, Editor: Arthur Chan

8.2 Before you start

8.2.1 The general-procedure chart

   [Editor Notes: Need to draw a nicer diagram]
8.2.2 Modeling context-dependent phones with untied states: some memory requirements

Modeling Context-dependent phones (ex. triphones) with untied states requires the largest amount of hardware resources. Take a moment to check if you have enough. The resources required depend on the type of model you are going to train, the dimensionality and configuration of your feature vectors, and the number of states in the HMMs.

Semi-continuous models

To train 5-state/HMM models for 10,000 triphones:

5 states/triphone = 50,000 states

For a 4-stream feature-set, each state has a total of 4*256 mixture weights = 1024 floating point numbers/state or (205Mb buffer for 50,000 states)

Corresponding to each of the four feature streams, there are 256 means and 256 variances in the codebook. ALL these, and ALL the mixture weights and transition matrices are loaded in into the RAM, and during training an additional buffer of equal size is allocated to store intermediate results. These are later written out into the hard disk when the calculations for the current training iteration are complete. Note that there are as many transition matrices as you have phones (40-50 for the English language, depending on your dictionary) All this amounts to allocating well over 400 Mb of RAM.

This is a bottleneck for machines with smaller memory. No matter how large your training corpus is, you can actually train only about 10,000 triphones at the cd-untied stage if you have 400 Mb of RAM (A 100 hour broadcast news corpus typically has 40,000 triphones). You could train more if your machine is capable of handling the memory demands effectively (this could be done, for example, by having a large amount of swap space). If you are training on multiple machines, *each* will require this much memory. In addition, at the end of each iteration, you have to transmit all buffers to a single machine that performs the norm. Networking issues need to be considered here.

The cd-untied models are used to build trees. The number of triphones you train at this stage directly affects the quality of the trees, which would have to be built using fewer triphones than are actually present in the training set if you do not have enough memory.
Continuous models

For 10,000 triphones:

5 states/triphone = 50,000 states

39 means (assuming a 39-component feature vector) and 39 variances per state = 79 floating points per state (or 15.8Mb buffer for 50,000 states)

Thus we can train 12 times as many triphones as we can when we have semicontinuous models for the same amount of memory. Since we can use more triphones to train (and hence more information) the decision trees are better, and eventually result in better recognition performance.

8.2.3 Data preparation

You will need the following files to begin training:

1. A set of feature files computed from the audio training data, one each for every recording you have in the training corpus. Each recording can be transformed into a sequence of feature vectors using a front-end executable provided with the SPHIN-III training package. Each front-end executable provided performs a different analysis of the speech signals and computes a different type of feature.

2. A control file containing the list of feature-set filenames with full paths to them. An example of the entries in this file:

   dir/subdir1/utt1
   dir/subdir1/utt2
   dir/subdir2/utt3

   Note that the extensions are not given. They will be provided separately to the trainer. It is a good idea to give unique names to all feature files, even if including the full paths seems to make each entry in the control file unique. You will find later that this provides a lot of flexibility for doing many things.

3. A transcript file in which the transcripts corresponding to the feature files are listed in exactly the same order as the feature filenames in the control file.

4. A main dictionary which has all acoustic events and words in the transcripts mapped onto the acoustic units you want to train. Redundancy in the form of extra words is permitted. The dictionary
must have all alternate pronunciations marked with parenthesized serial numbers starting from (2) for the second pronunciation. The marker (1) is omitted. Here’s an example:

DIRECTING  D A Y  R  E H  K  T  I  ng
DIRECTING(2)  D E R  E H  K  T  I  ng
DIRECTING(3)  D I R  E H  K  T  I  ng

5. A filler dictionary, which usually lists the non-speech events as ”words” and maps them to user-defined phones. This dictionary must at least have the entries

<s>  SIL
<sil>  SIL
</s>  SIL

6. The entries stand for

<s>: beginning-utterance silence
<sil>: within-utterance silence
</s>: end-utterance silence

Note that the words <s>, </s> and <sil> are treated as special words and are required to be present in the filler dictionary. At least one of these must be mapped on to a phone called ”SIL”. The phone SIL is treated in a special manner and is required to be present. The sphinx expects you to name the acoustic events corresponding to your general background condition as SIL. For clean speech these events may actually be silences, but for noisy speech these may be the most general kind of background noise that prevails in the database. Other noises can then be modelled by phones defined by the user.

During training SIL replaces every phone flanked by ”+” as the context for adjacent phones. The phones flanked by ”+” are only modeled as CI phones and are not used as contexts for triphones. If you do not want this to happen you may map your fillers to phones that are not flanked by ”+”.

7. A phonelist, which is a list of all acoustic units that you want to train models for. The SPHINX does not permit you to have units other than those in your dictionaries. All units in your two dictionaries must be listed here. In other words, your phonelist must have exactly the same units used in your dictionaries, no more and no less. Each phone must be listed on a separate line in the file, beginning from the left, with no extra spaces after the phone. an example:

   AA
Here’s a quick checklist to verify your data preparation before you train:

1. Are all the transcript words in the dictionary/filler dictionary?
2. Make sure that the size of transcript matches the .ctl file.
3. Check the boundaries defined in the .ctl file to make sure they exist i.e., you have all the frames that are listed in the control file
4. Verify the phonelist against the dictionary and fillerdic

**When you have a very small closed vocabulary (50-60 words)**

If you have only about 50-60 words in your vocabulary, and if your entire test data vocabulary is covered by the training data, then you are probably better off training word models rather than phone models. To do this, simply define the phoneset as your set of words themselves and have a dictionary that maps each word to itself and train. Also, use a lesser number of fillers, and if you do need to train phone models make sure that each of your tied states has enough counts (at least 5 or 10 instances of each).
8.2.4 The set of base and higher order feature vectors

The set of feature vectors you have computed using the Sphinx front-end executable is called the set of base feature vectors. This set of base features can be extended to include what are called higher order features. Some common extensions are

1. The set of difference vectors, where the component-wise difference between \( \text{*some*} \) succeeding and preceding vector(s), used to get an estimate of the slope or trend at the current time instant, are the "extension" of the current vector. These are called "delta" features. A more appropriate name would be the "trend" features.

2. The set of difference vectors of difference vectors. The component-wise difference between the succeeding and preceding "delta" vectors are the "extension" of the current vector. These are called "double delta" features

3. The set of difference vectors, where the component-wise difference between the \( n \)-th succeeding and \( n \)-th preceding vector are the "extension" of the current vector. These are called "long-term delta" features, differing from the "delta" features in just that they capture trends over a longer window of time.

4. The vector composed of the first elements of the current vector and the first elements of some of the above "extension" vectors. This is called the "power" feature, and its dimensionality is less than or equal to the total number of feature types you consider.

Feature streams

In semi-continuous models, it is a usual practice to keep the identities of the base vectors and their "extension" vectors separate. Each such set is called a "feature stream". You must specify how many feature streams you want to use in your semi-continuous models and how you want them arranged. The feature-set options currently supported by the Sphinx are:

\[
c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/: \text{read this as cepstra/second to last component, deltacepstra/second to last component, cepstra/first component deltacepstra/first component doubledeltacepstra/first component, doubledeltacepstra/second to last component}
\]

This is a 4-stream feature vector used mostly in semi-continuous models. There is no particular advantage to this arrangement - any permuta-
tion would give you the same models, with parameters written in different orders.

Here’s something that’s not obvious from the notation used for the 4-stream feature set: the dimensionality of the 4-stream feature vector is 12cepstra+24deltas+3powerterms+12doubledeltas

the deltas are computed as the difference between the cepstra two frames removed on either side of the current frame (12 of these), followed by the difference between the cepstra four frames removed on either side of the current frame (12 of these). The power stream uses the first component of the two-frames-removed deltas, computed using C0.

[Editor Notes: At this point, Rita stopped and said “(more to come....)” . Probably, we need to fill in some info.]

8.2.5 Force-alignment

FORCE-ALIGNMENT

Multiple pronunciations are not automatically considered in the SPHINX. You have to mark the right pronunciations in the transcripts and insert the interword silences. For this

1. remove the non-silence fillers from your filler dictionary and put them in your regular dictionary

2. Remove *all* silence markers (<s>, ¡sil¿ and </s>) from your training transcripts

For faligning with semi-continuous models, use the binary align provided with the trainer package with the following flag settings. For faligning with continuous models, change the settings of the flags -senmgaufn (.cont.), -topn (no. of Gaussians in the Gaussian mixture modeling each HMM state), -feat (the correct feature set):

- outsent
- insent
- ctlfn
- ctloffset 0
- ctlcount
- cepdir
- dict
8.3 Training Fully Continuous Hidden Markov Model

8.3.1 Creating the CI model definition file

The first step is to prepare a model definition file for the context independent (CI) phones. The function of a model definition file is to define or provide a unique numerical identity to every state of every HMM that you are going to train, and to provide an order which will be followed in writing out the model parameters in the model parameter files. During the training, the states are referenced only by these numbers. The model definition file thus partly specifies your model architecture and is thus usually stored in a directory named "model_architecture". You are of course free to store it where you please, unless you are running the training scripts provided with the SPHINX-III package.

To generate this CI model definition file, use the executable `mk_mdef.gen` with the following flag settings:

```plaintext
-fdict < filler dictionary >
-mdef
-senmgau .semi.
-mean
-var
-mixw
-tmat
-topn 4
-feat s2_4x
-beam 1e-90
-agc
-cmn
-logfn
```
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-phonelstfn</td>
<td>phonelist</td>
</tr>
<tr>
<td>-moddefn</td>
<td>name of the CI model definition file that you want to create. Full path must be provided</td>
</tr>
<tr>
<td>-n_state_pm</td>
<td>number of states per HMM in the models that you want to train. If you want to train 3 state HMMs, write &quot;3&quot; here, without the double quotes</td>
</tr>
</tbody>
</table>

Pipe the standard output into a log file `ci_mdef.log` (say). If you have listed only three phones in your phonelist, and specify that you want to build three state HMMs for each of these phones, then your model-definition file will look like this:

```
# Generated by mk_mdef_gen on Thu Aug 10 14:57:15 2000
0.3
3 n_base
0 n_tri
12 n_state_map
9 n_tied_state
9 n_tied_ci_state
3 n_tied_tmat
#
#
# Columns definitions
#base lft rt p attrib tmat ...state id’s ...
SIL - - - filler 0 0 1 2 N
A - - - n/a 1 3 4 5 N
B - - - n/a 2 6 7 8 N
```

The lines are simply comments.

The rest of the variables mean the following:

- `n_base`: no. of phones (also called "base" phones) that you have
- `n_tri`: no. of triphones (we will explain this later)
- `n_state_map`: Total no. of HMM states (emitting and non-emitting) The Sphinx appends an extra terminal non-emitting state to every HMM, hence for 3 phones, each specified by the user to be modeled by a 3-state HMM, this number will be 3phones*4states = 12
n_tied_state: no. of states of all phones after state-sharing is done. We
do not share states at this stage. Hence this number is the as the total
number of emitting states, $3 \times 3 = 9$

n_tied_ci_state: no. of states for your "base" phones after state-sharing
is done. At this stage, the number of "base" phones is the same as the
number of "all" phones that you are modeling. This number is thus again
the total number of emitting states, $3 \times 3 = 9$

n_tied_tmat: The HMM for each CI phone has a transition probability
matrix associated it. This is the total number of transition matrices for the
given set of models. In this case, this number is 3.

Columns definitions: The following columns are defined:

base : name of each phone
lft : left-context of the phone (- if none)
rt : right-context of the phone (- if none)
p : position of a triphone (not required at this stage)

attrib: attribute of phone. In the phone list, if the phone is "SIL",
or if the phone is enclosed by "+", as in "+BANG+", the sphinx under-
stands these phones to be non-speech events. These are also called "filler"
phones, and the attribute "filler" is assigned to each such phone. The
base phones have no special attributes, and hence are labelled as "n/a",
standing for "no attribute"

tmat : the id of the transition matrix associated with the phone

state id's : the ids of the HMM states associated with any phone. This
list is terminated by an "N" which stands for a non-emitting state. No id is
assigned to it. However, it exists, and is

8.3.2 Creating the HMM topology file

The HMM topology file consists of a matrix with boolean entries, each
entry indicates whether a specific transition from state=row_number to
state=column_number is permitted in the HMMs or not. For example a
3-state HMM with no skips permitted between states would have a topology
file with the following entries:

4

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>1.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td></td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

149
The number 4 is total the number of states in an HMMs. The SPHINX automatically appends a fourth non-emitting terminating state to the 3 state HMM. The first entry of 1.0 means that a transition from state 1 to state 1 (itself) is permitted. Accordingly, the transition matrix estimated for any phone would have a “transition-probability” in place of this boolean entry. Where the entry is 0.0, the corresponding transition probability will not be estimated (will be 0).

You can either write out the topology file manually, or use the script `make_topology.pl` provided with the SPHINX package to do this. The script needs the following arguments:

- `states_per_hmm`: this is merely an integer specifying the number of states per hmm
- `skipstate`: "yes" or "no" depending on whether you want the HMMs to have skipped state transitions or not.

Note that the topology file is common for all HMMs and is a single file containing the topology definition matrix. This file also defines your model architecture and is usually placed in the `model_architecture` directory. This is however optional, but recommended. If you are running scripts from the SPHINX training package, you will find the file created in the `model_architecture` directory.

### 8.3.3 Flat initialization of CI model parameters

CI models consist of 4 parameter files:

- `mixture_weights`: the weights given to every Gaussian in the Gaussian mixture corresponding to a state
- `transition_matrices`: the matrix of state transition probabilities
- `means`: means of all Gaussians
- `variances`: variances of all Gaussians
- `mdef`: model definition, the mapping from senone into states index.

To begin training the CI models, each of these files must have some initial entries, ie, they must be "initialized”. The mixture_weights and transition_matrices are initialized using the executable `mk_flat`. It needs the following arguments:
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-moddefn</td>
<td>CI model definition file</td>
</tr>
<tr>
<td>-topo</td>
<td>HMM topology file</td>
</tr>
<tr>
<td>-mixwfn</td>
<td>file in which you want to write the initialized \textit{mixture weights}</td>
</tr>
<tr>
<td>-tmatfn</td>
<td>file in which you want to write the initialized \textit{transition matrices}</td>
</tr>
<tr>
<td>-nstream</td>
<td>number of independent feature streams, for continuous models</td>
</tr>
<tr>
<td></td>
<td>this number should be set to &quot;1&quot;, without the double quotes</td>
</tr>
<tr>
<td>-ndensity</td>
<td>number of Gaussians modeling each state.</td>
</tr>
<tr>
<td></td>
<td>For CI models, this number should be set to &quot;1&quot;</td>
</tr>
</tbody>
</table>

To initialize the means and variances, global values of these parameters are first estimated and then copied into appropriate positions in the parameter files. The global mean is computed using all the vectors you have in your feature files. This is usually a very large number, so the job is divided into many parts. At this stage you tell the Sphinx how many parts you want it to divide this operation into (depending on the computing facilities you have) and the Sphinx "accumulates" or gathers up the vectors for each part separately and writes it into an intermediate buffer on your machine. The executable \texttt{init.gau} is used for this purpose. It needs the following arguments:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-accumdir</td>
<td>directory in which you want to write the intermediate buffers</td>
</tr>
<tr>
<td>-ctlfn</td>
<td>control file</td>
</tr>
<tr>
<td>-part</td>
<td>part number</td>
</tr>
<tr>
<td>-npart</td>
<td>total number of parts</td>
</tr>
<tr>
<td>-cepdir</td>
<td>path to feature files - this will be appended before all paths given in the control file</td>
</tr>
<tr>
<td>-ceplext</td>
<td>filename extension of feature files, eg. &quot;mfc&quot; for files called a/b/c.mfc. Double quotes are not needed</td>
</tr>
<tr>
<td>-feat</td>
<td>type of feature</td>
</tr>
<tr>
<td>-ceplen</td>
<td>dimensionality of base feature vectors</td>
</tr>
<tr>
<td>-agc</td>
<td>automatic gain control factor(max/none)</td>
</tr>
<tr>
<td>-cmn</td>
<td>cepstral mean normalization(yes/no)</td>
</tr>
<tr>
<td>-varnorm</td>
<td>variance normalization(yes/no)</td>
</tr>
</tbody>
</table>

Once the buffers are written, the contents of the buffers are "normalized" or used to compute a global mean value for the feature vectors. This is done using the executable \texttt{norm} with the following flag settings:
The next step is to "accumulate" the vectors for computing a global variance value. The executable **init gau**, when called a second time around, takes the value of the global mean and collects a set of (vector-globalmean) 2 values for the entire data set. This time around, this executable needs the following arguments:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-accumdir</td>
<td>directory in which you want to write the intermediate buffers</td>
</tr>
<tr>
<td>-meanfn</td>
<td>globalmean file</td>
</tr>
<tr>
<td>-ctlfn</td>
<td>control file</td>
</tr>
<tr>
<td>-part</td>
<td>part number</td>
</tr>
<tr>
<td>-npart</td>
<td>total number of parts</td>
</tr>
<tr>
<td>-cepdir</td>
<td>path to feature files - this will be appended before all paths given in the control file</td>
</tr>
<tr>
<td>-cepext</td>
<td>filename extension of feature files, eg. &quot;mfc&quot; for files called a/b/c.mfc. Double quotes are not needed</td>
</tr>
<tr>
<td>-feat</td>
<td>type of feature</td>
</tr>
<tr>
<td>-ceplen</td>
<td>dimensionality of base feature vectors</td>
</tr>
<tr>
<td>-agc</td>
<td>automatic gain control factor(max/none)</td>
</tr>
<tr>
<td>-cmn</td>
<td>cepstral mean normalization(yes/no)</td>
</tr>
<tr>
<td>-varnorm</td>
<td>variance normalization(yes/no)</td>
</tr>
</tbody>
</table>

Again, once the buffers are written, the contents of the buffers are "normalized" or used to compute a global variance value for the feature vectors. This is again done using the executable norm with the following flag settings:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-accumdir</td>
<td>buffer directory</td>
</tr>
<tr>
<td>-varfn</td>
<td>file in which you want to write the global variance</td>
</tr>
<tr>
<td>-feat</td>
<td>type of feature</td>
</tr>
<tr>
<td>-ceplen</td>
<td>dimensionality of base feature vector</td>
</tr>
</tbody>
</table>

Once the global mean and global variance are computed, they have to be copied into the means and variances of every state of each of the HMMs. The global mean is written into appropriate state positions in a means file while the global variance is written into appropriate state positions in a variances file. If you are using the scripts provided with the SPHINX package, you will find these files with "flatinitial" as part of its name in the
model parameters directory.

The flat means and variances file can be created using the executable cp.parm. In order to be able to use this executable you will have to create a copyoperations map file which is a two-column file, with the left column id-ing the state *to* which the global value has to be copied, and the right column id-ing the state *from* which it has to be copied. If there are "nphones" CI phones and each state has "n_state.per_hmm" EMITTING states, there will be ntotal_Estates = nphones * nEstate_per_hmm lines in the copyoperations map file; the state id-s (on the left column) run from 0 thru (ntotal_Estates - 1). Here is an example for a 3-state hmm (nEstate_per_hmm = 3) for two phones (nphones = 2) (ntotal_Estates = 6; so, state ids would vary from 0-5):

```
0 0
1 0
2 0
3 0
4 0
5 0
```

**cp.parm** requires the following arguments.

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-copropsfn</td>
<td>copyoperations map file</td>
</tr>
<tr>
<td>-igaufn</td>
<td>input global mean (or variance) file</td>
</tr>
<tr>
<td>-ncbouit</td>
<td>number of phones times the number of states per HMM</td>
</tr>
<tr>
<td></td>
<td>(ie, total number of states)</td>
</tr>
<tr>
<td>-ogaufn</td>
<td>output initialized means (or variances) file</td>
</tr>
</tbody>
</table>

**cp.parm** has to be run twice, once for copying the means, and once for copying the variances. This completes the initialization process for CI training.
8.3.4 Training CI models

Once the flat initialization is done, you are ready to begin training the acoustic models for the base or "context-independent" or CI phones. This step is called CI-training. In CI-training, the flat-initialized models are re-estimated through the forward-backward re-estimation algorithm called the Baum-Welch algorithm. This is an iterative re-estimation process, so you have to run many "passes" of the Baum-Welch re-estimation over your training data. Each of these passes, or iterations, results in a slightly better set of models for the CI phones. However, since the objective function maximized in each of these passes is the likelihood, too many iterations would ultimately result in models which fit very closely to the training data. You might not want this to happen for many reasons. Typically, 5-8 iterations of Baum-Welch are sufficient for getting good estimates of the CI models. You can automatically determine the number of iterations that you need by looking at the total likelihood of the training data at the end of the first iteration and deciding on a "convergence ratio" of likelihoods. This is simply the ratio of the total likelihood in the current iteration to that of the previous iteration. As the models get more and more fitted to the training data in each iteration, the training data likelihoods typically increase monotonically. The convergence ratio is therefore a small positive number. The convergence ratio becomes smaller and smaller as the iterations progress, since each time the current models are a little less different from the previous ones. Convergence ratios are data and task specific, but typical values at which you may stop the Baum-Welch iterations for your CI training may range from 0.1-0.001. When the models are variance-normalized, the convergence ratios are much smaller.

The executable used to run a Baum-Welch iteration is called "bw", and takes the following example arguments for training continuous CI models:
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-moddefn</td>
<td>model definition file for CI phones</td>
</tr>
<tr>
<td>-ts2cbfn</td>
<td>this flag should be set to &quot;.cont.&quot; if you are training continuous models, and to &quot;.semi.&quot; if you are training semi-continuous models, without the double quotes</td>
</tr>
<tr>
<td>-mixwfn</td>
<td>name of the file in which the mixture-weights from the previous iteration are stored. Full path must be provided</td>
</tr>
<tr>
<td>-mwfloor</td>
<td>Floor value for the mixture weights. Any number below the floor value is set to the floor value.</td>
</tr>
<tr>
<td>-tmatfn</td>
<td>name of the file in which the transition matrices from the previous iteration are stored. Full path must be provided</td>
</tr>
<tr>
<td>-meanfn</td>
<td>name of the file in which the means from the previous iteration are stored. Full path must be provided</td>
</tr>
<tr>
<td>-varfn</td>
<td>name of the file in which the variances from the previous iteration are stored. Full path must be provided</td>
</tr>
<tr>
<td>-dictfn</td>
<td>Dictionary</td>
</tr>
<tr>
<td>-fdictfn</td>
<td>Filler dictionary</td>
</tr>
<tr>
<td>-ctlfn</td>
<td>control file</td>
</tr>
<tr>
<td>-part</td>
<td>You can split the training into N equal parts by setting a flag. If there are M utterances in your control file, then this will enable you to run the training separately on each (M/N)th part. This flag may be set to specify which of these parts you want to currently train on. As an example, if your total number of parts is 3, this flag can take one of the values 1, 2 or 3</td>
</tr>
<tr>
<td>-npart</td>
<td>number of parts in which you have split the training</td>
</tr>
<tr>
<td>-cepidir</td>
<td>directory where your feature files are stored</td>
</tr>
<tr>
<td>-cepext</td>
<td>the extension that comes after the name listed in the control file. For example, you may have a file called a/b/c.d and may have listed a/b/c in your control file. Then this flag must be given the argument &quot;d&quot;, without the double quotes or the dot before it</td>
</tr>
<tr>
<td>-lsnfn</td>
<td>name of the transcript file</td>
</tr>
<tr>
<td>-accumdir</td>
<td>Intermediate results from each part of your training will be written in this directory. If you have T means to estimate, then the size of the mean buffer from the current part of your training will be T*4 bytes (say). There will likewise be a variance buffer, a buffer for mixture weights, and a buffer for transition matrices</td>
</tr>
<tr>
<td>-varfloor</td>
<td>minimum variance value allowed</td>
</tr>
<tr>
<td>-topn</td>
<td>no. of gaussians to consider for computing the likelihood of each state. For example, if you have 8 gaussians/state models and topn is 4, then the 4 most likely gaussian are used.</td>
</tr>
<tr>
<td>-abeam</td>
<td>forward beamwidth</td>
</tr>
<tr>
<td>-bbeam</td>
<td>backward beamwidth</td>
</tr>
<tr>
<td>-agc</td>
<td>automatic gain control</td>
</tr>
<tr>
<td>-cmn</td>
<td>cepstral mean normalization</td>
</tr>
<tr>
<td>-varnorm</td>
<td>variance normalization</td>
</tr>
<tr>
<td>-meanreest</td>
<td>mean re-estimation</td>
</tr>
<tr>
<td>-varreest</td>
<td>variance re-estimation</td>
</tr>
</tbody>
</table>
If you have run the training in many parts, or even if you have run the training in one part, the executable for Baum-Welch described above generates only intermediate buffer(s). The final model parameters, namely the means, variances, mixture-weights and transition matrices, have to be estimated using the values stored in these buffers. This is done by the executable called "norm", which takes the following arguments:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-accumdir</td>
<td>Intermediate buffer directory</td>
</tr>
<tr>
<td>-feat</td>
<td>feature configuration</td>
</tr>
<tr>
<td>-mixwfn</td>
<td>name of the file in which you want to write the mixture weights. Full path must be provided</td>
</tr>
<tr>
<td>-tmatfn</td>
<td>name of the file in which you want to write the transition matrices. Full path must be provided</td>
</tr>
<tr>
<td>-meanfn</td>
<td>name of the file in which you want to write the means. Full path must be provided</td>
</tr>
<tr>
<td>-varfn</td>
<td>name of the file in which you want to write the variances. Full path must be provided</td>
</tr>
<tr>
<td>-ceplen</td>
<td>length of basic feature vector</td>
</tr>
</tbody>
</table>

If you have not re-estimated any of the model parameters in the bw step, then the corresponding flag must be omitted from the argument given to the norm executable. The executable will otherwise try to read a non-existent buffer from the buffer directory and will not go through. Thus if you have set -meanreest to be "no" in the argument for bw, then the flag -meanfn must not be given in the argument for norm. This is useful mostly during adaptation.

Iterations of baum-welch and norm finally result CI models. The iterations can be stopped once the likelihood on the training data converges. The model parameters computed by norm in the final iteration are now used to initialize the models for context-dependent phones (triphones) with untied states. This is the next major step of the training process. We refer to the process of training triphones HMMs with untied states as the "CD untied training".
8.3.5 Creating the CD untied model definition file

The next step is the CD-untied training, in which HMMs are trained for all context-dependent phones (usually triphones) that are seen in the training corpus. For the CD-untied training, we first need to generate a model definition file for all the triphones occurring in the training set. This is done in several steps:

First, a list of all triphones possible in the vocabulary is generated from the dictionary. To get this complete list of triphones from the dictionary, it is first necessary to write the list of phones in the following format:

```
phone1 0 0 0 0
phone2 0 0 0 0
phone3 0 0 0 0
phone4 0 0 0 0
...```

The phonelist used for the CI training must be used to generate this, and the order in which the phones are listed must be the same.

Next, a temporary dictionary is generated, which has all words except the filler words (words enclosed in ++()++). The entry

```
SIL  SIL```

must be added to this temporary dictionary, and the dictionary must be sorted in alphabetical order. The program `quick.count` provided with the SPHINX-III package can now be used to generate the list of all possible triphones from the temporary dictionary. It takes the following arguments:

<table>
<thead>
<tr>
<th>FLAG DESCRIPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>mandatory flag to tell <code>quick.count</code> to consider all word pairs while constructing triphone list</td>
</tr>
<tr>
<td>-p</td>
<td>formatted phonelist</td>
</tr>
<tr>
<td>-b</td>
<td>temporary dictionary</td>
</tr>
<tr>
<td>-o</td>
<td>output triphone list</td>
</tr>
</tbody>
</table>

Here is a typical output from `quick.count`

```
AA(AA, AA) s 1
AA(AA, AE) b 1
AA(AA, AO) l 1
AA(AA, AW) e 1```

The "1" in AA(AA,AO)l indicates that this is a word-internal triphone. This is a carry over from Sphinx-II. The output from `quick.count` has to be now written into the following format:

```
AA AA AA s```
This can be done by simply replacing "(" , ",", and ")" in the output of quick.count by a space and printing only the first four columns. While doing so, all instances of " 1" must be replaced by " i". To the top of the resulting file the list of CI phones must be appened in the following format

```
AA - - -
AE - - -
AO - - -
AW - - -
.
.
AA AA AA s
AA AA AE b
AA AA AO i
AA AA AW e
```

For example, if the output of the quick.count is stored in a file named "quick.count.out", the following perl command will generate the phone list in the desired form.

```
[ Editor Notes: Should put back the perl command here. ]
```

The above list of triphones (and phones) is converted to the model definition file that lists all possible triphones from the dictionary. The program used from this is "mk_mdef_gen" with the following arguments number of states per HMM

```
FLAG DESCRIPTION -moddeffn model definition file with all possible triphones(alltriphones.mdef) to be written -phonelstfn list of all triphones -n_state.pm
```

In the next step we find the number of times each of the tripho
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-moddefn</td>
<td>model definition file with all possible triphones(alltriphones.mdef)</td>
</tr>
<tr>
<td>-ts2cbfn</td>
<td>takes the value “.cont.” if you are building continuous models</td>
</tr>
<tr>
<td>-ctfln</td>
<td>control file corresponding to your training transcripts</td>
</tr>
<tr>
<td>-lsnfn</td>
<td>transcript file for training</td>
</tr>
<tr>
<td>-dictfn</td>
<td>training dictionary</td>
</tr>
<tr>
<td>-fdictfn</td>
<td>filler dictionary</td>
</tr>
<tr>
<td>-paramtype</td>
<td>write “phone” here, without the double quotes</td>
</tr>
<tr>
<td>-segdir</td>
<td>/dev/null</td>
</tr>
</tbody>
</table>

**param_cnt** writes out the counts for each of the triphones onto stdout. All other messages are sent to stderr. The stdout therefore has to be directed into a file. If you are using csh or tcsh it would be done in the following manner:

```
[ Editor Notes: add param_cnt here]
```

Here’s an example of the output of this program

```
+GARBAGE+   -   -   98
+LAUGH+     -   -   29
SIL         -   -   31694
AA          -   -   0
AE          -   -   0
  
  
  
  
AA AA AA  s  1
AA AA AE  s  0
AA AA AO  s  4
```

The final number in each row shows the number of times that particular triphone (or filler phone) has occurred in the training corpus. Note that if all possible triphones of a CI phone are listed in the all_triphones.mdef the CI phone itself will have 0 counts since all instances of it would have been mapped to a triphone.

This list of counted triphones is used to shortlist the triphones that have occurred a minimum number (threshold) of times. The shortlisted triphones appear in the same format as the file from which they have been selected. The shortlisted triphone list has the same format as the triphone list used to generate the all_triphones.mdef. The formatted list of CI phones has to be included in this as before. So, in the earlier example, if a threshold of 4 were used, we would obtain the shortlisted triphone list as
The threshold is adjusted such that the total number of triphones above the threshold is less than the maximum number of triphones that the system can train (or that you wish to train). It is good to train as many triphones as possible. The maximum number of triphones may however be dependent on the memory available on your machine. The logistics related to this are described in the beginning of this manual.

Note that thresholding is usually done so to reduce the number of triphones, in order that the resulting models will be small enough to fit in the computer's memory. If this is not a problem, then the threshold can be set to a smaller number. If the triphone occurs too few times, however, (ie, if the threshold is too small), there will not be enough data to train the HMM state distributions properly. This would lead to poorly estimated CD untied models, which in turn may affect the decision trees which are to be built using these models in the next major step of the training.

A model definition file is now created to include only these shortlisted triphones. This is the final model definition file to be used for the CD untied training. The reduced triphone list is then to the model definition file using **mk_mdef_gen** with the following arguments: number of states per HMM

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-moddefn</td>
<td>model definition file for CD untied training</td>
</tr>
<tr>
<td>-phonelstfn</td>
<td>list of shortlisted triphones</td>
</tr>
<tr>
<td>-n_state_pm</td>
<td></td>
</tr>
</tbody>
</table>

Finally, therefore, a model definition file which lists all CI phones and seen triphones is constructed. This file, like the CI model-definition file, assigns unique id's to each HMM state and serves as a reference file for handling and identifying the CD-untied model parameters. Here is an example of the CD-untied model-definition file: If you have listed five phones in your phones.list file,

SIL B AE T

160
and specify that you want to build three state HMMs for each of these phones, and if you have one utterance listed in your transcript file:

<s> BAT A TAB </s> for which your dictionary and fillerdict entries are:

Fillerdict:

<s> SIL </s>

</s> SIL

Dictionary:

A AX

BAT B AE T

TAB T AE B

then your CD-untied model-definition file will look like this:

[ Editor Notes: Model definition should be attached to here.]

8.3.6 Flat initialization of CD untied model parameters

FLAT INITIALIZATION OF CD UNTIED MODEL PARAMETERS

In the next step in CD untied training, after the CD untied model definition file has been constructed, the model parameters are first initialized. During this process, the model parameter files corresponding to the CD untied model-definition file are generated. Four files are generated: means, variances, transition matrices and mixture weights. In each of these files, the values are first copied from the corresponding CI model parameter file. Each state of a particular CI phone contributes to the same state of the same CI phone in the Cd -untied model parameter file, and also to the same state of the *all* the triphones of the same CI phone in the CD-untied model parameter file. The CD-untied model definition file is of course used as a reference for this mapping. This process, as usual, is called "initialization".

Initialization for the CD-untied training is done by the executable called init_mixw. It need the following arguments:
8.3.7 Training CD untied models

Once the initialization for CD-untied training is done, the next step is to actually train the CD untied models. To do this, as in the CI training, the Baum-Welch forward-backward algorithm is iteratively used. Each iteration consists of generating bw buffers by running the bw executable on the training corpus (this can be divided into many parts as explained in the CI training), followed by running the norm executable to compute the final parameters at the end of the iteration.

The arguments of `bw` and `norm` could be found in ??

[ Editor Notes: There are usually couple of precautions in training we need to mention, what are they?]
Generating the linguistic questions

The decision trees require the CD-untied models and a set of predefined phonetic classes (or classes of the acoustic units you are modeling) which share some common property. These classes or questions are used to partition the data at any given node of a tree. Each question results in one partition, and the question that results in the “best” partition (maximum increase in likelihood due to the partition) is chosen to partition the data at that node. All linguistic questions are written in a single file called the “linguistic questions” file. One decision tree is built for each state of each phone.

For example, if you want to build a decision tree for the contexts (D B P AE M IY AX OY) for any phone, then you could ask the question: does the context belong to the class vowels? If you have defined the class vowels to have the phones AE AX IY OY EY AA EH (in other words, if one of your linguistic questions has the name “VOWELS” and has the elements AE AX IY OY EY AA EH corresponding to that name), then the decision tree would branch as follows:

[Editor Notes: Draw a diagram on decision tree]

Here is an example of a "linguistic-questions" file:

```
ASPSEG HH
SIL SIL
VOWELS AE AX IY OY EY AA EH
ALVSTP D T N
DENTAL DH TH
LABSTP B P M
LIQUID L R
```

The column on the left specifies the name given to the class. This name is user defined. The classes consist of a single phone or a cluster of phones which share some common acoustic property. If your acoustic units are not completely phonetically motivated, or if you are training models for a language whose phonetic structure you are not completely sure about, then the executable classed `make quests` provided with the SPHINX-III package can be used to generate the linguistic questions. It uses the CI models to make the questions, and needs the following arguments:
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-moddeffn</td>
<td>CI model definition file</td>
</tr>
<tr>
<td>-meanfn</td>
<td>CI means file</td>
</tr>
<tr>
<td>-varfn</td>
<td>CI variances file</td>
</tr>
<tr>
<td>-mixwfn</td>
<td>CI mixture weights file</td>
</tr>
<tr>
<td>-npermute</td>
<td>A bottom-up top-down clustering algorithm is used to group the phones into classes. Phones are clustered using bottom-up clustering until npermute classes are obtained. The npermute classes are exhaustively partitioned into two classes and evaluated to identify the optimal partitioning of the entire phone set into two groups. An identical procedure is performed recursively on each of these groups to generate an entire tree. npermute is typically between 8 and 12. Smaller values of npermute result in suboptimal clustering. Larger values become computationally prohibitive.</td>
</tr>
<tr>
<td>-niter</td>
<td>The bottom-up top-down clustering can be iterated to give more optimal clusters. niter sets the number of iterations to run. niter is typically set to 1 or 2. The clustering saturates after 2 iterations.</td>
</tr>
<tr>
<td>-qstpersst</td>
<td>The algorithm clusters state distributions belonging to each state of the CI phone HMMs to generate questions. Thus all 1st states are clustered to generate one subset of questions, all 2nd states are clustered for the second subset, and so on. qstpersst determines how many questions are to be generated by clustering any state. Typically, this is set to a number between 20 and 25.</td>
</tr>
<tr>
<td>-tempfn</td>
<td></td>
</tr>
<tr>
<td>-questfn</td>
<td>output linguistic questions file</td>
</tr>
</tbody>
</table>

Once the linguistic questions have been generated, decision trees must be built for each state of each CI phone present in your phonelist. Decision trees are however not built for filler phones written as +0+ in your phonelist. They are also not built for the SIL phone. In order to build decision trees, the executable “bldtree” must be used. It takes the following arguments:
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-treefn</td>
<td>full path to the directory in which you want the decision trees to be written</td>
</tr>
<tr>
<td>-moddeffn</td>
<td>CD-untied model definition file</td>
</tr>
<tr>
<td>-mixwfn</td>
<td>CD-untied mixture weights file</td>
</tr>
<tr>
<td>-ts2cbfn</td>
<td>.cont.</td>
</tr>
<tr>
<td>-meanfn</td>
<td>CD-untied means file</td>
</tr>
<tr>
<td>-varfn</td>
<td>CD-untied variances file</td>
</tr>
<tr>
<td>-mwfloor</td>
<td>Floor value of the mixture weights. Values below this are reset to this value. A typical value is 1e-8</td>
</tr>
<tr>
<td>-psetfn</td>
<td>linguistic questions file</td>
</tr>
<tr>
<td>-phone</td>
<td>CI phone for which you want to build the decision tree</td>
</tr>
<tr>
<td>-state</td>
<td>The HMM state for which you want to build the decision tree. For a three state HMM, this value can be 0,1 or 2. For a 5 state HMM, this value can be 0,1,2,3 or 4, and so on</td>
</tr>
<tr>
<td>-stwt</td>
<td>This flag needs a string of numbers equal to the number of HMM-states, for example, if you were using 5-state HMMs, then the flag could be given as &quot;-stwt 1.0 0.3 0.1 0.01 0.001&quot;. Each of these numbers specify the weights to be given to state distributions during tree building, beginning with the <em>current</em> state. The second number specifies the weight to be given to the states <em>immediately adjacent</em> to the current state (if there are any), the third number specifies the weight to be given to adjacent states <em>one removed</em> from the immediately adjacent one (if there are any), and so on. A typical set of values for 5 state HMMs is &quot;1.0 0.3 0.1 0.01 0.001&quot;</td>
</tr>
<tr>
<td>-ssplitmin</td>
<td>Complex questions are built for the decision tree by first building &quot;pre-trees&quot; using the linguistic questions in the question file. The minimum number of bifurcations in this tree is given by ssplitmin. This should not be lesser than 1. This value is typically set to 1.</td>
</tr>
<tr>
<td>-ssplitmax</td>
<td>The maximum number of bifurcations in the simple tree before it is used to build complex questions. This number is typically set to 7. Larger values would be more computationally intensive. This number should not be smaller than the value given for ssplitmin</td>
</tr>
<tr>
<td>-ssplitthr</td>
<td>Minimum increase in likelihood to be considered for a bifurcation in the simple tree. Typically set to a very small number greater than or equal to 0</td>
</tr>
<tr>
<td>-csplitmin</td>
<td>The minimum number of bifurcations in the decision tree. This should not be less than 1</td>
</tr>
<tr>
<td>-csplitmax</td>
<td>The maximum number of bifurcations in the decision tree. This should be as large as computationally feasible. This is typically set to 2000</td>
</tr>
<tr>
<td>-csplitthr</td>
<td>Minimum increase in likelihood to be considered for a bifurcation in the decision tree. Typically set to a very small number greater than or equal to 0</td>
</tr>
<tr>
<td>-cntthresh</td>
<td>Minimum number of observations in a state for it to be considered in the decision tree building process</td>
</tr>
</tbody>
</table>
If, for example, you have a phonelist which contains the following phones
+NOISE+
SIL
AA
AX
B

and you are training 3 state HMMs, then you must build 9 decision
trees, one each for each state of the phones AA, AX and B.
8.3.9 Pruning the decision trees

PRUNING THE DECISION TREES

Once the decision trees are built, they must be pruned to have as many leaves as the number of tied states (senones) that you want to train. Remember that the number of tied states does not include the CI states, which are never tied. In the pruning process, the bifurcations in the decision trees which resulted in the minimum increase in likelihood are progressively removed and replaced by the parent node. The selection of the branches to be pruned out is done across the entire collection of decision trees globally. The executable to be used for decision tree pruning is called "prunetree" and requires the following arguments:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-itreedir</td>
<td>directory in which the full decision trees are stored</td>
</tr>
<tr>
<td>-nseno</td>
<td>number of senones that you want to train</td>
</tr>
<tr>
<td>-otreedir</td>
<td>directory to store the pruned decision trees</td>
</tr>
<tr>
<td>-moddeffn</td>
<td>CD-untied model definition file</td>
</tr>
<tr>
<td>-psetfn</td>
<td>linguistic questions file</td>
</tr>
<tr>
<td>-minocc</td>
<td>minimum number of observations in the given tied state. If there are fewer observations, the branches corresponding to the tied state get pruned out by default. This value should never be 0, otherwise you will end up having senones with no data to train (which are seen 0 times in the training set)</td>
</tr>
</tbody>
</table>

8.3.10 Creating the CD tied model definition file

Once the trees are pruned, a new model definition file must be created which

- contains all the triphones which are seen during training
- has the states corresponding to these triphones identified with senones from the pruned trees

In order to do this, the model definition file which contains all possible triphones from the current training dictionary can be used [alltriphones model definition file]. This was built during the process of building the CD-untied model definition file. Remember that the CD-untied model definition file contained only a selected number of triphones, with various thresholds used for selection. That file, therefore, cannot be used to build the CD-tied model definition file, except in the exceptional case where you are sure that the CD-untied model definition file includes *all* triphones
seen during training. The executable that must be used to tie states is called “tiesate” and needs the following arguments:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-imoddeffn</td>
<td>alltriphones model definition file</td>
</tr>
<tr>
<td>-omoddeffn</td>
<td>CD-tied model definition file</td>
</tr>
<tr>
<td>-treedir</td>
<td>pruned tree directory</td>
</tr>
<tr>
<td>-psetfn</td>
<td>linguistic questions file</td>
</tr>
</tbody>
</table>

Here is an example of a CD-tied model definition file, based on the earlier example given for the CD-untied model definition file. The alltriphones model definition file:

```
# Generated by [path]/mk_mdef_gen on Sun Nov 26 12:42:05 2000
# triphone:  (null)
# seno map:  (null)
#
0.3
5 n_base
34 n_tri
156 n_state_map
117 n_tied_state
15 n_tied_ci_state
5 n_tied_tmat
#
# Columns definitions
#base lft rt p attrib tmat ... state id's ...
SIL -- -- filler 0 0 1 2 N
AE -- -- n/a 1 3 4 5 N
AX -- -- n/a 2 6 7 8 N
B -- -- n/a 3 9 10 11 N
T -- -- n/a 4 12 13 14 N
AE B T i n/a 1 15 16 17 N
AE T B i n/a 1 18 19 20 N
AX AX AX s n/a 2 21 22 23 N
AX AX B s n/a 2 24 25 26 N
```
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX AX SIL s n/a 2 27 28 29 N</td>
<td>AX AX T s n/a 2 30 31 32 N</td>
<td>AX B AX s n/a 2 33 34 35 N</td>
<td>AX B B s n/a 2 36 37 38 N</td>
<td>AX B SIL s n/a 2 39 40 41 N</td>
<td>AX B T s n/a 2 42 43 44 N</td>
</tr>
<tr>
<td>AX SIL AX s n/a 2 45 46 47 N</td>
<td>AX SIL B s n/a 2 48 49 50 N</td>
<td>AX SIL SIL s n/a 2 51 52 53 N</td>
<td>AX SIL T s n/a 2 54 55 56 N</td>
<td>AX T AX s n/a 2 57 58 59 N</td>
<td></td>
</tr>
<tr>
<td>AX T B s n/a 2 60 61 62 N</td>
<td>AX T SIL s n/a 2 63 64 65 N</td>
<td>AX T T s n/a 2 66 67 68 N</td>
<td>B AE AX e n/a 3 69 70 71 N</td>
<td>B AE B e n/a 3 72 73 74 N</td>
<td></td>
</tr>
<tr>
<td>B AE SIL e n/a 3 75 76 77 N</td>
<td>B AE T e n/a 3 78 79 80 N</td>
<td>B AX AE b n/a 3 81 82 83 N</td>
<td>B B AE b n/a 3 84 85 86 N</td>
<td>B SIL AE b n/a 3 87 88 89 N</td>
<td></td>
</tr>
<tr>
<td>B T AE b n/a 3 90 91 92 N</td>
<td>T AE AX e n/a 4 93 94 95 N</td>
<td>T AE B e n/a 4 96 97 98 N</td>
<td>T AE SIL e n/a 4 99 100 101 N</td>
<td>T AE T e n/a 4 102 103 104 N</td>
<td></td>
</tr>
<tr>
<td>T AX AE b n/a 4 105 106 107 N</td>
<td>T B AE b n/a 4 108 109 110 N</td>
<td>T SIL AE b n/a 4 111 112 113 N</td>
<td>T T AE b n/a 4 114 115 116 N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
is used as the base to give the following CD-tied model definition file with 39 tied states (senones):

```plaintext
# Generated by [path]/mk_mdef_gen on Sun Nov 26 12:42:05 2000
# triphone:  (null)
# seno map:  (null)
#
0.3
5 n_base
34 n_tri
156 n_state_map
54 n_tied_state
15 n_tied_ci_state
5 n_tied_tmat
#
# Columns definitions
#base lft rt p attrib tmat ... state id’s ...
SIL - - - filler 0 0 1 2 N
AE - - - n/a 1 3 4 5 N
AX - - - n/a 2 6 7 8 N
B - - - n/a 3 9 10 11 N
T - - - n/a 4 12 13 14 N
AE B T i n/a 1 15 16 17 N
AE T B i n/a 1 18 16 19 N
AX AX AX s n/a 2 20 21 22 N
AX AX B s n/a 2 23 21 22 N
AX AX SIL s n/a 2 24 21 22 N
AX AX T s n/a 2 25 21 22 N
AX B AX s n/a 2 26 21 27 N
AX B B s n/a 2 23 21 27 N
AX B SIL s n/a 2 24 21 27 N
AX B T s n/a 2 25 21 27 N
```

170
<table>
<thead>
<tr>
<th>AX SIL AX</th>
<th>s</th>
<th>n/a 2 26 21 28 N</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX SIL B</td>
<td>s</td>
<td>n/a 2 23 21 28 N</td>
</tr>
<tr>
<td>AX SIL SIL</td>
<td>s</td>
<td>n/a 2 24 21 28 N</td>
</tr>
<tr>
<td>AX SIL T</td>
<td>s</td>
<td>n/a 2 25 21 28 N</td>
</tr>
<tr>
<td>AX T AX</td>
<td>s</td>
<td>n/a 2 26 21 29 N</td>
</tr>
<tr>
<td>AX T B</td>
<td>s</td>
<td>n/a 2 23 21 29 N</td>
</tr>
<tr>
<td>AX T SIL</td>
<td>s</td>
<td>n/a 2 24 21 29 N</td>
</tr>
<tr>
<td>AX T T</td>
<td>s</td>
<td>n/a 2 25 21 29 N</td>
</tr>
<tr>
<td>B AE AX</td>
<td>e</td>
<td>n/a 3 30 31 32 N</td>
</tr>
<tr>
<td>B AE B</td>
<td>e</td>
<td>n/a 3 33 31 32 N</td>
</tr>
<tr>
<td>B AE SIL</td>
<td>e</td>
<td>n/a 3 34 31 32 N</td>
</tr>
<tr>
<td>B AE T</td>
<td>e</td>
<td>n/a 3 35 31 32 N</td>
</tr>
<tr>
<td>B AX AE</td>
<td>b</td>
<td>n/a 3 36 37 38 N</td>
</tr>
<tr>
<td>B B AE</td>
<td>b</td>
<td>n/a 3 36 37 39 N</td>
</tr>
<tr>
<td>B SIL AE</td>
<td>b</td>
<td>n/a 3 36 37 40 N</td>
</tr>
<tr>
<td>B T AE</td>
<td>b</td>
<td>n/a 3 36 37 41 N</td>
</tr>
<tr>
<td>T AE AX</td>
<td>e</td>
<td>n/a 4 42 43 44 N</td>
</tr>
<tr>
<td>T AE B</td>
<td>e</td>
<td>n/a 4 45 43 44 N</td>
</tr>
<tr>
<td>T AE SIL</td>
<td>e</td>
<td>n/a 4 46 43 44 N</td>
</tr>
<tr>
<td>T AE T</td>
<td>e</td>
<td>n/a 4 47 43 44 N</td>
</tr>
<tr>
<td>T AX AE</td>
<td>b</td>
<td>n/a 4 48 49 50 N</td>
</tr>
<tr>
<td>T B AE</td>
<td>b</td>
<td>n/a 4 48 49 51 N</td>
</tr>
<tr>
<td>T SIL AE</td>
<td>b</td>
<td>n/a 4 48 49 52 N</td>
</tr>
<tr>
<td>T T AE</td>
<td>b</td>
<td>n/a 4 48 49 53 N</td>
</tr>
</tbody>
</table>
8.3.11 Initializing and training cd tied gaussian mixture models

The next step is to train the CD-tied models. In the case of continuous models, the HMM states can be modeled by either a single Gaussian distribution, or a mixture of Gaussian distributions. The number of Gaussians in a mixture-distribution must preferably be even, and a power of two (for example, 2, 4, 8, 16, 32, ...). To model the HMM states by a mixture of 8 Gaussians (say), we first have to train 1 Gaussian per state models. Each Gaussian distribution is then split into two by perturbing its mean slightly, and the resulting two distributions are used to initialize the training for 2 Gaussians per state models. These are further perturbed to initialize for 4 Gaussians per state models and a further split is done to initialize for the 8 Gaussians per state models. So the CD-tied training for models with 2N Gaussians per state is done in N+1 steps. Each of these N+1 steps consists of

1. initialization

2. iterations of Baum-Welch followed by norm

3. Gaussian splitting (not done in the N+1th stage of CD-tied training)

The training begins with the initialization of the 1 Gaussian per state models. During initialization, the model parameters from the CI model parameter files are copied into appropriate positions in the CD tied model parameter files. Four model parameter files are created, one each for the means, variances, transition matrices and mixture weights. During initialization, each state of a particular CI phone contributes to the same state of the same CI phone in the CD-tied model parameter file, and also to the same state of the *all* the triphones of the same CI phone in the CD-tied model parameter file. The CD-tied model definition file is used as a reference for this mapping.

Initialization for the 1 gaussian per state models

[Editor Notes: Put an example of init_mixw at here]

To split mixture, you just need to use the command **inc_comp**.

The arguments needed by **inc_comp** are:
<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ninc</td>
<td>how many gaussians (per state) to split currently. You need not always split to double the number of Gaussians. You can specify other numbers here, so long as they are less than the number of Gaussians you currently have. This is a positive integer like &quot;2&quot;, given without the double quotes</td>
</tr>
<tr>
<td>-ceplen</td>
<td>length of the base feature vector</td>
</tr>
<tr>
<td>-dcountfn</td>
<td>input mixture weights file</td>
</tr>
<tr>
<td>-inmixwfn</td>
<td>input mixture weights file</td>
</tr>
<tr>
<td>-outmixwfn</td>
<td>output mixture weights file</td>
</tr>
<tr>
<td>-inmeanfn</td>
<td>input means file</td>
</tr>
<tr>
<td>-outmeanfn</td>
<td>output means file</td>
</tr>
<tr>
<td>-invarfn</td>
<td>input variances file</td>
</tr>
<tr>
<td>-outvarfn</td>
<td>output variances file</td>
</tr>
<tr>
<td>-feat</td>
<td>type of feature</td>
</tr>
</tbody>
</table>
8.4 Training semi-continuous models

8.4.1 Difference between training continuous HMM and semi-continuous HMM

[Editor Notes: Wrap up with one utterance to say that these process is the same as CD]
8.4.2 Vector quantization

This is done in two steps. In the first step, the feature vectors are accumulated for quantizing the vector space. Not all feature vectors are used. Rather, a sampling of the vectors available is done by the executable "aggseg". This executable simply "aggregates" the vectors into a buffer. The following flag settings must be used with this executable:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-segdmpdirs</td>
<td>directory in which you want to put the aggregate buffer</td>
</tr>
<tr>
<td>-segdmpfn</td>
<td>name of the buffer (file)</td>
</tr>
<tr>
<td>-segtype</td>
<td>all</td>
</tr>
<tr>
<td>-ctlfn</td>
<td>control file</td>
</tr>
<tr>
<td>-cepdir</td>
<td>path to feature files</td>
</tr>
<tr>
<td>-cepext</td>
<td>feature vector filename extension</td>
</tr>
<tr>
<td>-ceplen</td>
<td>dimensionality of the base feature vector</td>
</tr>
<tr>
<td>-agc</td>
<td>automatic gain control factor(max/none)</td>
</tr>
<tr>
<td>-cmn</td>
<td>cepstral mean normalization(yes/no)</td>
</tr>
<tr>
<td>-feat</td>
<td>type of feature. As mentioned earlier, the 4-stream feature vector is usually given as an option here. When you specify the 4-stream feature, this program will compute and aggregate vectors corresponding to all streams separately.</td>
</tr>
<tr>
<td>-stride</td>
<td>how many samples to ignore during sampling of vectors (pick every stride’th sample)</td>
</tr>
</tbody>
</table>

In the second step of vector quantization, an Expectation-Maximization (EM) algorithm is applied to segregate each aggregated stream of vectors into a codebook of N Gaussians. Usually N is some power of 2, the commonly used number is N=256. The number 256 can in principle be varied, but this option is not provided in the SPHINX-II decoder. So if you intend to use the SPHINX-II decoder, but are training models with SPHINX-III trainer, you must use N=256. It has been observed that the quality of the models built with 256 codeword codebooks is sufficient for good recognition. Increasing the number of codewords may cause data-insufficiency problems. In many instances, the choice to train semi-continuous models (rather than continuous ones) arises from insufficiency of training data. When this is indeed the case, increasing the number of codebooks might aggravate the estimation problems that might arise due to data insufficiency. Consider this fact seriously before you decide to increase N.

In SPHINX-III, the EM-step is done through a k-means algorithm carried out by the executable kmeansinit. This executable is usually used with the following flag settings:

- grandvar yes
-gthobj single
-stride 1
-ntrial 1
-minratio 0.001
-ndensity 256
-meanfn full_path_to_codebookmeans.file
-varfn full_path_to_codebookvariances.file
-reest no
-segdmpdirs directory_in_which_you_want_to_put_aggregate.file
-segdmpfn aggregate.file
-ceplen dimensionality_of_feature_vector
-feat type_of_feature
-agc automatic_gain_control_factor(max/none)
-cmn cepstral_mean_normalization(yes/no)

Once the vector quantization is done, you have to flat-initialize your acoustic models to prepare for the first real step in training. The following steps explain the flat-initialization process:
Deleted interpolation is the final step in creating semi-continuous models. The output of deleted interpolation are semi-continuous models in sphinx-3 format. These have to be further converted to sphinx-2 format, if you want to use the SPHINX-II decoder.

Deleted interpolation is an iterative process to interpolate between CD and CI mixture-weights to reduce the effects of overfitting. The data are divided into two sets, and the data from one set are used to estimate the optimal interpolation factor between CI and CD models trained from the other set. Then the two data sets are switched and this procedure is repeated using the last estimated interpolation factor as an initialization for the current step. The switching is continued until the interpolation factor converges.

To do this, we need *two* balanced data sets. Instead of the actual data, however, we use the Baum-Welch buffers, since the related math is convenient. we therefore need an *even* number of buffers that can be grouped into two sets. DI cannot be performed if you train using only one buffer. At least in the final iteration of the training, you must perform the training in (at least) two parts. You could also do this serially as one final iteration of training AFTER BW has converegd, on a non-lsf setup.

Note here that the norm executable used at the end of every Baum-Welch iteration also computes models from the buffers, but it does not require an even number of buffers. BW returns numerator terms and denominator terms for the final estimation, and norm performs the actual division. The number of buffers is not important, but you would need to run norm at the end of EVERY iteration of BW, even if you did the training in only one part. When you have multiple parts norm sums up the numerator terms from the various buffers, and the denominator terms, and then does the division.

The executable "delint" provided with the SPHINX-III package does the deleted interpolation. It takes the following arguments:
8.5 Using SCHMM's decision trees to train a FCHMM

[Editor Notes: This part not yet finished]

After the decision trees are built using semi-continuous models, it is possible to train continuous models. ci-semicontinuous models need to be trained for initializing the semicontinuous untied models. ci-continuous models need to be trained for initializing the continuous tied state models. the feature set can be changed after the decision tree building stage.

8.6 Updating or adapting existing models sets

In general one is better off training speaker specific models if sufficient data (at least 8-10 hours) are available. If you have less data for a speaker or a domain, then the better option is to adapt any existing models you have to the data. Exactly how you adapt would depend on the kind of acoustic models you’re using. If you’re using semi-continuous models, adaptation could be performed by interpolating speaker specific models with speaker-independent models. For continuous HMMs you would have to use MLLR, or one of its variants. To adapt or update existing semicontinuous models, follow these steps:

[Editor Notes: We need an example here......]

1. Compute features for the new training data. The features must be computed in the same manner as your old training features. In fact, the feature computation in the two cases must be identical as far as possible.
2. Prepare transcripts and dictionary for the new data. The dictionary must have the same phoneset as was used for training the models. The transcripts must also be prepared in the same manner. If you have new filler phones then the fillerdict must map them to the old filler phones.

3. The new training transcript and the corresponding ctl file can include the old training data IF all you are doing is using additional data from the SAME domain that you might have recently acquired. If you are adapting to a slightly different domain or slightly different acoustic conditions, then use only the new data.

4. Starting with the existing deleted-interpolated models, and using the same tied mdef file used for training the base models and the same training parameters like the difference features, number of streams etc., run through one or two passes of Baum-Welch. However, this must be done without re-estimating the means and variances. Only the mixture-weights must be re-estimated. If you are running the norm after the Baum-Welch, then make sure that the norm executable is set to normalize only the mixture weights.

5. Once the mixture weights are re-estimated, the new mixture weights must be interpolated with the ones you started with. The executable ”mixw_interp” provided with the SPHINX package may be used for this. You can experiment with various mixing weights to select the optimal one. This is of course the simplest update/adaptation technique. There are more sophisticated techniques which will be explained here later.

The **mixw_interp** executable:

This is used in model adaptation for interpolating between two mixture weight files. It requires the following flags:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-SImixwfn</td>
<td>The original Speaker-Independent mixture weights file</td>
</tr>
<tr>
<td>-SDmixwfn</td>
<td>The Speaker Dependent mixture weight file that you have after the bw iterations for adaptation</td>
</tr>
<tr>
<td>-tokencntfn</td>
<td>The token count file</td>
</tr>
<tr>
<td>-outmixwfn</td>
<td>The output interpolated mixture weight parameter file name</td>
</tr>
<tr>
<td>-SIlambda</td>
<td>Weight given to SI mixing weights</td>
</tr>
</tbody>
</table>
8.7 Training Multilingual Models

Once you have acoustic data and the corresponding transcriptions for any language, and a lexicon which translates words used in the transcription into sub-word units (or just maps them into some reasonable-looking acoustic units), you can use the SPHINX to train acoustic models for that language. You do not need anything else.

The linguistic questions that are needed for building the decision trees are automatically designed by the SPHINX. Given the acoustic units you choose to model, the SPHINX can automatically determine the best combinations of these units to compose the questions. The hybrid algorithm that the SPHINX uses clusters state distributions of context-independent phones to obtain questions for triphonetic contexts. This is very useful if you want to train models for languages whose phonetic structure you do not know well enough to design your own phone classes (or if a phonetician is not available to help you do it). An even greater advantage comes from the fact that the algorithm can be effectively used in situations where the subword units are not phonetically motivated. Hence you can comfortably use any set of acoustic units that look reasonable to you for the task.

If you are completely lost about the acoustic units but have enough training data for all (or most) words used in the transcripts, then build word models instead of subword models. You do not have to build decision trees. Word models are usually context-independent models, so you only have to follow through the CI training. Word models do have some limitations, which are currently discussed in the non-technical version of this manual.

[ Editor Notes: This part needs to greatly expand.]

8.8 The training lexicon

Inconsistencies in the training lexicon can result in bad acoustic models. Inconsistencies stem from the usage of a phoneset with phones that are confusible in the pattern space of our recognizer. To get an idea about the confusibility of the phones that you are using, look at the per-frame log likelihoods of the utterances during training. A greater number of phones in the lexicon should ordinarily result in higher log likelihoods. If you have a baseline to compare with, and this is *not* the case, then it means that the phoneset is more diffuse over the pattern space (more compact, if you observe the opposite for a smaller phone set), and the corresponding distributions are wider (sharper in the other case). Generally, as the number
of applicable distributions decreases over a given utterance, the variances tend to become larger and larger. The distributions flatten out since the areas under the distributions are individually conserved (to unity) and so the overall per frame likelihoods are expected to be lower.

The solution is to fix the phoneset, and to redo the lexicon in terms of a phoneset of smaller size covering the acoustic space in a more compact manner. One way to do this is to collapse the lexicon into syllables and longer units and to expand it again using a changed and smaller phoneset. The best way to do this is still a research problem, but if you are a native speaker of the language and have a good ear for sounds, your intuition will probably work. The SPHINX will, of course, be able to train models for any new phoneset you come up with.

8.9 10 Common Pitfalls of Using SphinxTrain

Author: Arthur Chan, Editor: Arthur Chan

At here, I include some common pitfalls (10 of them) in using Sphinx-Train. They are not very technical. It is basically an expansion of Rita’s Sphinx manual’s Section 1, ”Before you train” So I won’t touch issues such as usage of individual commands and such. I just want to give a big picture of what is training using SphinxTrain in general.

Pitfall I: Incorrect expectation of the training process

Acoustic model training is a very involved process. As far as I could say, this is true for HTK and SphinxTrain. Though both HTK and SphinxTrain has fairly detail documentation. It is still pretty difficult to go through one complete it. Though, I want to say I love this experience. :-)

There are several things you need to be prepared before you go through the training process. I will list it here.

1. Large amount of training data. This training data usually appear in the form of so-called corpus. Or you need to collect them.

2. Computation time. You need a computer, preferably a fast one, if you want to complete the training task in a reasonable time.

3. Expertise of the process and willing to understand what’s going on. You will find that if you knew nothing of the process, you probably cannot survive. :-)

181
4. patience of the human trainer. Even with automated script, training can easily go wrong. Debugging usually takes a lot of effort and carelessness is always punished in a sinister way. :-) 

In the next several pitfalls, I will talk about more in detail each of the above points.

**Pitfall II: Incorrect expectation of the effort of training (II)**

Usage of the trainer is very different from the decoder. Training is a very different process from decoding. Most people get frustrated in training. Mainly because they could use the decoder but found that the training system is harder to use.

**Pitfall III: Confusion of different models in Sphinxen**

About models, there are two main types of models used by Sphinxen. In the past, Sphinx 2 only supported only SCHMM. Sphinx 4 and Sphinx 3.x decode only supported only CDHMM. Sphinx 3.0 family of tools (e.g. decode_anytopo, align, allphone) supported both SCHMM and CDHMM. Rita’s sphinxtrain manual has very detail summary of what’s going on how to train both SCHMM and CDHMM. These formatting tends to change, e.g. In 2004, Sphinx 2 starts to support CDHMM as well.

**Pitfall IV: Move on without reading manual**

This is a point I always want to stress. Did you check out Rita’s manual on how to use SphinxTrain? Most of the content of the manual is still correct at this point. The only thing which has been changed is probably the fact that we using perl script to automate the job.

**Pitfall V: Misunderstanding about the tools and scripts in SphinxTrain**

Scripts of SphinxTrain, when it was designed, was mainly used by researcher. So you will find most problems you faced can be quickly solved by inspecting the script and see what’s going on.

Why don’t we write a very thorough package and so that the user complexity is lower? As a matter of fact, we did, but surprisingly it caused a lot of problems internal to CMU. Training is a process where a lot of parts
Pitfall VI: Lack of Training Data

Do you have enough training data? For every gaussian in a senone, you probably need 100 samples of frames.

Small system such as TIDIGTS, each HMM were trained by at least 100 waveforms. If you want mixture models to be effective, you need even more.

Usually a medium system (such as RM) need 10 hours of speech to train. Most successful large vocabulary system used more than 50 hours of speech to train. Some even more if a larger user coverage was to be achieved.

Data need to be collected in a balanced way, that means you could not collect a lot of samples for 1 type of sentence but only a little for another. This simply hurts the system.

Pitfall VII: Incorrect expectation of computation time

Baum-Welch algorithm is fairly time-consuming. SphinxTrain’s algorithm is optimized in a way that the time required is significantly reduced. However, training a model with 80 hours of data still required approximately 200 machine hours. (For a P4 1.1G)

In most of user’s cases, their training set is significantly smaller. However, the effect of training time still overwhelm many people (literally). I heard a lot of people giving up training just because they don’t like to wait for a long time. This is unfortunately not the expectation one should have with acoustic model training.

Ask yourself again. Do you have enough computation time? Usually, using 1 machine, small vocabulary system such as TIDIGITS could take you few hours. RM take probably half a day. A lot of things I am working on take 2 to 3 weeks if there is not parallelization of the script.
Pitfall VIII: Incorrect expectation of expertise.

Do you have enough expertise? I will judge that acoustic model training requires at least a bright undergraduate student to do.

There are a lot of things you need to learn in training and in general speech recognition. HMM, signal processing, language modeling. None of them are trivial. Rabiner’s book on speech recognition is a must for starter. XD Huang’s "Spoken Language Processing" is necessary for you to become an expert.

What if I could not afford this expensive books? I would recommend you some more accessible source. For example:

"A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". This is a short version of Prof. Rabiner’s book. You can get a very good idea of what’s going on with speech recognition. It used discrete HMM but the general idea is the same.

This might sound very scary but hey! Don’t worry :-), you can always learn when you work on SphinxTrain. It is actually a satisfying process. I feel pretty happy myself when I work out an acoustic model. (Feeling similar to cooking.) I also learn a lot in this process.

Though remember this: no pain, no gain. I saw a lot of Sphinx’ users just want to do something to complete their homework or term project. They even don’t have the motivation to learn what’s going on in speech recognition. I usually consider these cases are helpless.

Pitfall IX: Incorrect expectation of the task completion time.

Do you have enough patience? For patience I mean patience in learning, patience in waiting and patience in debugging.

Using the trainer is harder than using the decoder. This is a universal characteristic for all systems including HTK, Sphinx (I used them both, I also wrote some training algorithm myselfs). What I found is if you have patience to learn, most issues will be solved by yourself.

I say this because I saw a lot of you sending to Sphinx’s forums just because you were stopped by a small problem. Or a lot of times many people promised their boss that they could train a system in 1 day. You better be prepared because no serious speech people will think in this way.
Pitfall X: Ineffective questions in Sphinx's Forum

This is generally not a SphinxTrain specific question. This is actually very general problem that happen in every Sphinx's family of software.

In general, it makes a lot of sense to ask questions in this forum. I also love to answer users' question because I can learn a lot from them. Though I found that certain netiquette will make all our experience to be more pleasant.

1. Read the manual before asking questions.
2. Send the output to the list, instead of questions like "I got a problem.".
3. Send your mail to one single forum
4. Try to meditate the conversation yourself, if you start to digress, it is not a bad idea to start to new topic. Other users will find your thread more readable

The document is actually there and it is pretty well-organized in cmusphinx.org. Things which are harder to find usually mean we have no intention to open it or we haven’t tested it thoroughly. We chose to do it because we genuinely hope that our code can go to the user’s hands as soon as possible.

Though I have to admit here for one thing. Not everything in Sphinx can be described as perfect. Despite our continuous effort to improve it, there are still a lot of aspect I personally feel it could be even better. Documentation is one of those. Multiple authors have been contributing and inconsistency is somehow unavoidable. Our effort in merging the manual is always going on. Of course, this is not something that could be done in one to two day.
Chapter 9

Language Model Training

Author: Roni Rosenfeld and Philip Clarkson, Editor: Arthur Chan

[ Editor Notes: Plain copy from Philip's web site. Pretty dangerous to just open source it. ]

9.1 Installation of the Toolkit

For "big-endian" machines (eg those running HP-UX, IRIX, SunOS, Solaris) the installation procedure is simply to change into the src/ directory and type

$ make install

The executables will then be copied into the bin/ directory, and the library file SLM2.a will be copied into the lib/ directory. For "little-endian" machines (eg those running Ultrix, Linux) the variable BYTESWAP_FLAG will need to be set in the Makefile. This can be done by editing src/Makefile directly, so that the line

#BYTESWAP_FLAG = -DSLM_SWAP_BYTES

is changed to

BYTESWAP_FLAG = -DSLM_SWAP_BYTES

Then the program can be installed as before.

If you are unsure of the "endian-ness" of your machine, then the shell script endian.sh should be able to provide some assistance.
In case of problems, then more information can be found by examining src/Makefile.

Before building the executables, it might be worth adjusting the value of STD_MEM in the file src/toolkit.h. This value controls the default amount of memory (in MB) that the programs will attempt to assign for the large buffers used by some of the programs (this value can, of course, be overridden at the command line). The result is that the final process sizes will be a few MB bigger than this value. The more memory that can be grabbed, the faster the programs will run. The default value is 100, but if the machines which the tools will be run on contain less, or much more memory than this, then this value should be adjusted to reflect this.
### 9.2 Terminology and File Formats

<table>
<thead>
<tr>
<th><strong>Text stream</strong></th>
<th>An ASCII file containing text. It may or may not have markers to indicate context cues, and white space can be used freely.</th>
<th>.text</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Word frequency file</strong></td>
<td>An ASCII file containing a list of words, and the number of times that they occurred. This list is not sorted; it will generally be used as the input to wfreq2vocab, which does not require sorted input.</td>
<td>.wfreq</td>
</tr>
<tr>
<td><strong>Word n-gram file</strong></td>
<td>ASCII file containing an alphabetically sorted list of n-tuples of words, along with the number of occurrences</td>
<td>.w3gram, .w4gram etc.</td>
</tr>
<tr>
<td><strong>Vocabulary file</strong></td>
<td>ASCII file containing a list of vocabulary words. Comments may also be included - any line beginning ## is considered a comment. The vocabulary is limited in size to 65535 words.</td>
<td>.vocab.20K, .vocab.60K etc., depending on the size of the vocabulary.</td>
</tr>
<tr>
<td><strong>Context cues file</strong></td>
<td>ASCII file containing the list of words which are to be considered &quot;context cues&quot;. These are words which provide useful context information for the n-grams, but which are not to be predicted by the language model. Typical examples would be ¡s¿ and ¡p¿, the begin sentence, and begin paragraph tags.</td>
<td>.ccs</td>
</tr>
<tr>
<td><strong>Id n-gram file</strong></td>
<td>ASCII or binary (by default) file containing a numerically sorted list of n-tuples of numbers, corresponding to the mapping of the word n-grams relative to the vocabulary. Out of vocabulary (OOV) words are mapped to the number 0. .id3gram.bin, .id4gram.ascii etc. Binary language model file Binary file containing all the n-gram counts, together with discounting information and back-off weights. Can be read by evallm and used to generate word probabilities quickly.</td>
<td>.binlm</td>
</tr>
<tr>
<td><strong>ARPA language model file</strong></td>
<td>ASCII file containing the language model probabilities in ARPA-standard format.</td>
<td>.arpa</td>
</tr>
<tr>
<td><strong>Probability stream</strong></td>
<td>ASCII file containing a list of probabilities (one per line). The probabilities correspond the the probability for each word in a specific text stream, with context-cues and OOVs removed.</td>
<td>.fprobs</td>
</tr>
<tr>
<td><strong>Forced back-off file</strong></td>
<td>ASCII file containing a list of vocabulary words from which to enforce back-off, together with either an 'i' or an 'e' to indicate inclusive or exclusive forced back-off respectively.</td>
<td>.fblist</td>
</tr>
</tbody>
</table>
These files may all be written are read by all the tools in compressed or uncompressed mode. Specifically, if a filename is given a .Z extension, then it will be read from the specified file via a zcat pipe, or written via a compress pipe. If a filename is given a .gz, it will be read from the specified file via a gunzip pipe, or written via a gzip pipe. If either of these compression schemes are to be used, then the relevant tools (ie zcat, and compress or gzip) must be available on the system, and pointed to by the path.

If a filename argument is given as - then it is assumed to represent either the standard input, or standard output (according to context). Any file read from the standard input is assumed to be uncompressed, and therefore, all desired compression and decompression should take place in a pipe:

```
$ zcat < abc.Z | abc2xyz | compress > xyz.Z
```

### 9.3 Typical Usage

Given a large corpus of text in a file a.text, but no specified vocabulary.

- Compute the word unigram counts
  
  ```
  $ cat a.text | text2wfreq > a.wfreq
  ```

- Convert the word unigram counts into a vocabulary consisting of the 20,000 most common words
  
  ```
  $ cat a.wfreq | wfreq2vocab -top 20000 > a.vocab
  ```

- Generate a binary id 3-gram of the training text, based on this vocabulary
  
  ```
  $ cat a.text | text2idngram -vocab a.vocab > a.idngram
  ```

- Convert the idngram into a binary format language model
  
  ```
  $ idngram2lm -idngram a.idngram -vocab a.vocab -binary a.binlm
  ```

- Compute the perplexity of the language model, with respect to some test text b.text
  
  ```
  evallm -binary a.binlm
  ```
  
  Reading in language model from file a.binlm
  
  Done.
  
  ```
  evallm : perplexity -text b.text
  ```
Computing perplexity of the language model with respect to the text b.text
Perplexity = 128.15, Entropy = 7.00 bits
Computation based on 8842804 words.
Number of 3-grams hit = 6806674 (76.97%)
Number of 2-grams hit = 1766798 (19.98%)
Number of 1-grams hit = 269332 (3.05%)
1218322 OOVs (12.11%) and 576763 context cues were removed from the calculation.

evallm : quit

Alternatively, some of these processes can be piped together:

cat a.text | text2wfreq | wfreq2vocab -top 20000 > a.vocab
cat a.text | text2idngram -vocab a.vocab |
           idngram2lm -vocab a.vocab -idngram -
           -binary a.binlm -spec.num 5000000 15000000
echo "perplexity -text b.text" | evallm -binary a.binlm

9.4 Discounting Strategies

Discounting is the process of replacing the original counts with modified counts so as to redistribute the probability mass from the more commonly observed events to the less frequent and unseen events. If the actual number of occurrences of an event $E$ (such as a bigram or trigram occurrence) is $c(E)$, then the modified count is $d(c(E))c(E)$, where $d(c(E))$ is known as the discount ratio.

9.4.1 Good Turing discounting

Good Turing discounting defines $d(r) = \frac{(r+1)n(r+1)}{rn(r)}$ where $n(r)$ is the number of events which occur $r$ times.

The discounting is only applied to counts which occur fewer than $K$ times, where typically $K$ is chosen to be around 7. This is the "discounting range" which is specified using the -disc_ranges parameter of the idngram2lm program.

9.4.2 Witten Bell discounting

The discounting scheme which we refer to here as "Witten Bell discounting" is that which is referred to as type C in "The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression", Ian H. Witten and Timothy C. Bell, in "IEEE Transactions on Information Theory, Vol 37, No. 4, July 1991".

The discounting ratio is not dependent on the event’s count, but on \( t \), the number of types which followed the particular context. It defines \( d(r, t) = \frac{n}{n+t} \), where \( n \) is the size of the training set in words. This is equivalent to setting \( P(w|h) = \frac{c}{n+t} \) (where \( w \) is a word, \( h \) is the history and \( c \) is the number of occurrences of \( w \) in the context \( h \)), for events that have been seen, and \( P(w|h) = \frac{t}{n+t} \) for unseen events. Absolute discounting

Absolute discounting defines \( d(r) = \frac{(r-b)}{r} \). Typically \( b = \frac{n(1)}{(n(1)+2n(2))} \). The discounting is applied to all counts.

This is, of course, equivalent to simply subtracting the constant \( b \) from each count.

9.4.3 Linear discounting

Linear discounting defines \( d(r) = 1 - \frac{n(1)}{C} \), where \( C \) is the total number of events. The discounting is applied to all counts.

Chapter 10

Search structure and Speed-up of the speech recognizer

10.1 Introduction

Author: Arthur Chan, Ravi Mosur, Editor: Arthur Chan

[Editor Notes: This is largely adapted from Ravi Mosur's code-walk document file.]

In this chapter, we will describe the search structure of the recognizers in Sphinx. There are several recognizers in the standard Sphinx 3’s distribution package. Each of them can be used for certain particular purpose. It is very important to understand their differences and apply each of them correctly. For example, s3.X decode_anytopo in 2004 was designed to be used as an offline batch mode recognizer. It is designed to be accurate and also optimization was not applied on the recognizer. These are all done by design. If you accidentally used this recognizer for real-time application, I am sure you will lose your client. :-}
10.2 **Sphinx 3X’s recognizer general architecture**

Conceptually, all recognizers in Sphinx 3X share the same recognition engine. One can always view Sphinx’s search as two separated but interrelated components.

1. GMM Computation module

2. Graph Search module

At every frame, GMM computation will compute all the active GMM that marked by the graph search module. The graph search module will make use of the GMM scores to do the graph searching. Since there is pruning mechanism in the search. Only part of the GMM will be found to be active in the next frame. The information on whether the GMM is active will be used by GMM computation module.

In a first pass system, this separation of GMM computation and search modules are found to be very useful. The major reason is that techniques can be applied to individual modules and speed performance can be benchmarked separately. This usually provides valuable insights for further improvement of the system.

The history of the speech recognizer’s development shows that there are two major inter-related goals of speech recognizer design. One is to optimize the accuracy of speech recognizer. One is to allow an known imperfect speech recognizer to be used in a practical scenario. Usually, this practical scenario impose constraints to the design of the recognizer such that the recognizer has to run in a limited amount of computation resource (such as amount of random access memory and the computation power). For example, NIST evaluation, a kind of competition between research sites, requires each site to provide the best performing recognition system. This basically requires users to fulfill the first goal. Usually, these systems are run-in around 20x-100xRT.

Sphinx, follow this guide, were implemented under similar historical trend, the Sphinx 3.0 recognizer (or s3slow, s3 accurate) are designed to be an accurate but perhaps slow recognizer. In 2004, running S3 on the Wall Street Journal 5000 Words Task require 10-12xRT of computation.\(^1\) This is the most accurate among the Sphinx’s family of recognizers. Whereas, Sphinx 3.X (or s3fast) can run the task in 0.94xRT with around 5% relative degradation.

\(^1\)5000 senones, trigram
It is therefore understandable that why two implementations of the speech recognition engine were happened in the first place. s3slow is always meant for research purpose and allow the most accurate results. s3fast is always meant to be designed as a fast and perhaps application specific recognizer. Hence, one found only small amount of speed optimization technique appears in s3slow. However, one can see s3fast accept employ different techniques in general and allow the user to choose them.

There exists customized modifications of each of the recognizers. For example, Arthur Chan has written a version of s3 which employ fast GMM computation. This was unfortunately not suitable to be incorporated into the main trunk of the s3slow. One can get this information from sphinx developer’s web page

www.cs.cmu.edu/~archan/

The conceptual model described about is not fixed and from time to time, we will revise it and possibly incorporate more interesting features to the recognizers.

10.3 Importance of tuning

It is widespread misunderstanding that a single default generic speech models can be used in all different situations. It is also incorrect to assume that the default parameters used in speech recognition is the best in general. First-class speech recognition system was tuned intensively for both speed and accuracy. In this chapter, apart from just describing the principle of search. We will also describe parameters that can affect the operation of the parameters. Internal to CMU, this parameter was tuned through exhaustively trial and error testing using a small test set.

So how about you? We recommend you to read the following description before you start to do tuning. Usually, tuning is done by tuning one parameter at a time. It is also possible to device a kind of multi-dimensional search is possible for this kind of problem.

10.4 Sphinx 3.X’s decode anytopo (or s3slow)

In one sentence, decode anytopo is a generic tri-gram speech recognizer which accept the use of multi-stream continuous HMM with tied mixture
and with any topologies of the HMM. At 2004 Sep, you can get this recognizer at

```
$ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/cmusphinx co archive_s3/
```

What it means is that one can use the standard fully continuous HMM (1-stream, tied state) or semi-continuous HMM (4-stream, tied mixture) as an input of the model. The user is allowed to specify the grammar as the form N-gram, It is very generic and can be used in most of the situation.

We will describe several aspects of decode_anytopo in this chapter. Note that decode_anytopo doesn’t contain too much optimization in the code as we already mentioned its purpose is more on benchmarking.

### 10.4.1 GMM Computation

Gaussian Mixture Model (GMM or senone in CMU’s terminology) was computed as it is, usually only active GMM is computed at every frame. One special aspect of GMM computation is that its output is a scaled integer. Back to 1995, this is the only way that log values can be efficiently computed. Internally, the code is also optimized using loop-unrolling. Other than these two techniques, that is no approximation at all.

### 10.4.2 Search structure

The data structure for including different words in search is arranged as flat lexicons. This is the standard way to arrange the lexicon and will not have bad effect on accuracy as the tree lexicon (as described in Section ?). However, not able to tied the prefixes of different words together will double the time of traversing lattice. This is part of the reason why decode_anytopo is at least 2 times slower than decode.

The score accumulation in the search is based on integer. Again, the reason is that computing the log value of a number is best to be done using lookup tables when the recognizer is first implemented. Even, till now, computing log value using standard C library is still not as fast as integer lookup table.

---

2Loop unrolling is a technique in C-programming that makes try to take advantage of parellel scheduling in modern day central processor. It is a common way to optimize code in a for loop
10.4.3 Treatment of language model

decode_anytopo can take care of trigram in a single-pass search. However, exhaustive search for full-trigram is extremely expansive because it always requires storing the two previous words at a particular time. It will actually caused the storage require in search to become $n^2$ instead of $n$. Instead of doing so, Sphinx 3 used so called it poor man trigram in decoding. The idea is that for a given word $w_n$, only store one single previous word $w_{n-1}$. For $w_{n-2}$, only used the most likely previous word that derived from the Viterbi search. In early experiment back to the date of evaluation of Wall Street Journal (WSJ) task (93-96). This assumption is found to be a very good approximation (less than 5% degradation of accuracy). It was therefore employed and used until nowadays.

10.4.4 Triphone representation

[Under construction]

10.4.5 Viteri Pruning

Although decode_anytopo is designed to be used for evaluation, it also implemented standard pruning feature in the search. Major reason is that people long realized that fully exhaustive Viterbi search will search a lot of very unlikely paths. Viterbi Pruning is the simplest and the most effective way to reduce the number of paths search.

There are two beams used in decodeanytopo,

- **-beam** Main pruning beam applied to triphones in forward search
- **-nwbeam** Pruning beam applied in forward search upon word exit

10.4.6 Search Tuning

There are several important parameters in the search that can affect the accuracy.

- **-langwt** Language weight: empirical exponent applied to LM probaby
• **-ugwt** LM unigram weight: unigram probs interpolated with unifodistribution with this weight

• **-inspen** Word insertion penalty

• **-silpen** Language model 'probability' of silence word

• **-noisepen** Language model 'probability' of each non-silence filler d

• **-fillpenfn** Filler word probabilities input file (used in place of -pen and -noisepen)

### 10.4.7 2-nd pass Search

One can used decode_anytopo to become a building block of what so called the multi-pass system. This is based on generating a word lattice of hypothesis instead of generating a single hypothesis. Then rescoring is done based on the lattice. The options that control this properties are,

• **-bestpath** Whether to run bestpath DAG search after forward Viterss

• **-dagfudge** (0..2); 1 or 2: add edge if endframe == startframe; 2:tart == end-1

• **-bestpathlw** Language weight for bestpath DAG search (default: same-langwt)

• **-inlatdir** Input word-lattice directory with per-utt files for reting words searched

• **-inlatwin** Input word-lattice words starting within +/- ὲthis argγ of current frame considered during search

• **-outlatdir** Directory for writing word lattices (one file/utterance optional .NODES suffix to write only the nodes

• **-latext** Word-lattice filename extension (.gz or .Z extension fmpression)

• **-bestscoredir** Directory for writing best score/frame (used to set best; one file/utterance)

We will describe the behavior of 2-nd pass search in the Section ?.
10.4.8 Debugging

There are many situations you would like to get internal information of the Sphinx 3.0’s recognizer.

- **-hmmdumpsf** Starting frame for dumping all active HMMs (for debugdiagnosis/analysis)
- **-worddumpsf** Starting frame for dumping all active words (for debug-diagnosis/analysis)
- **-logfn** Log file (default stdout/stderr)
- **-backtrace** Whether detailed backtrace information (word segmentat-
cores) shown in log

10.5 Sphinx 3.X’s decode (aka s3 fast)

The so called Sphinx 3.X’s decode or s3fast was implemented 4 years after implementation of decode_anytopo. The major motivation in those days was to allow a recognizer can be ran in less than 10xRT for the Broadcast News task. This target is fullfilled in those days and it is estimated that with proper tuning, broadcast news may be able to performed under less than 1xRT in within the next 5 years even using single-pass search. ³

At 2004, it will be safe to assume that the 1-pass search can be tuned to speed less than 1xRT for system with less than 10000 words with LM perplexity around 80.

The major reason why it can be much faster than decode_anytopo is because many techniques have been applied to speed-up the search. This section will describe each of these techniques in detail.

10.6 Architecture of Search in decode

Author: Ravi Mosur and Arthur Chan, Editor: Arthur Chan

³Writing at 2004 Sep, at that time, a PC equipped with 3G Pentium can produce 3-
4xRT performance in s3.X
10.6.1 Initialization

The decoder is configured during the initialization step, and the configuration holds for the entire run. This means, for example, that the decoder does not dynamically reconfigure the acoustic models to adapt to the input. To choose another example, there is no mechanism in this decoder to switch language models from utterance to utterance, unlike in Sphinx-II. The main initialization steps are outlined below.

Log-Base Initialization. Sphinx performs all likelihood computations in the log-domain. Furthermore, for computational efficiency, the base of the logarithm is chosen such that the likelihoods can be maintained as 32-bit integer values. Thus, all the scores reported by the decoder are log-likelihood values in this peculiar log-base. The default base is typically 1.0003, and can be changed using the -logbase configuration argument. The main reason for modifying the log-base would be to control the length (duration) of an input utterance before the accumulated log-likelihood values overflow the 32-bit representation, causing the decoder to fail catastrophically. The log-base can be changed over a wide range without affecting the recognition.

Models Initialization: The lexical, acoustic, and language models specified via the configuration arguments are loaded during initialization. This set of models is used to decode all the utterances in the input. (The language model is actually only partly loaded, since s3.X uses a disk-based LM strategy. Though it is also possible to load the LM into memory by setting -lminmemory to 1)

Effective Vocabulary: After the models are loaded, the effective vocabulary is determined. It is the set of words that the decoder is capable of recognizing. Recall that the decoder is initialized with three sources of words: the main and filler lexicon files, and the language model. The effective vocabulary is determined from them as follows:

- Find the intersection of the words in the LM and the main pronunciation lexicon
- Include all the alternative pronunciations to the set derived above (using the main lexicon
- Include all the filler words from the filler lexicon, but excluding the distinguished beginning and end of sentence words: <s> and </s>.

The effective vocabulary remains in effect throughout the batch run. Currently, it is not possible to add to or remove from this vocabulary dynamically, unlike in the Sphinx-II system.
A pronunciation lexicon (or dictionary) file specifies word pronunciations. In Sphinx, pronunciations are specified as a linear sequence of phonemes. Each line in the file contains one pronunciation specification, except that any line that begins with a "#" character in the first column is treated as a comment and is ignored. Example dictionary for digits:

```
ZERO Z IH R OW
ONE W AH N
TWO T UW
THREE TH R IY
FOUR F AO R
FIVE F AY V
SIX S IH K S
SEVEN S EH V AX N
EIGHT EY TD
NINE N AY N
```

The lexicon is completely case-insensitive (unfortunately). For example, it's not possible to have two different entries Brown and brown in the dictionary. Multiple Pronunciations

A word may have more than one pronunciation, each one on a separate line. They are distinguished by a unique parenthesized suffix for the word string. For example:

```
ACTUALLY AE K CH AX W AX L IY
ACTUALLY(2nd) AE K SH AX L IY
ACTUALLY(3rd) AE K SH L IY
```

If a word has more than one pronunciation, its first appearance must be the unparenthesized form. For the rest, the parenthesized suffix may be any string, as long as it is unique for that word. There is no other significance to the order of the alternatives; each one is considered to be equally likely. Compound Words

In Sphinx-3, the lexicon may also contain compound words. A compound word is usually a short phrase whose pronunciation happens to differ significantly from the mere concatenation of the pronunciations of its constituent words. Compound word tokens are formed by concatenating the component word strings with an underscore character; e.g.:
The s3.X decoder, however, treats a compound word as just another word in the language, and does not do anything special with it.)

Tree representation

The decoder constructs lexical trees from the effective vocabulary described above. Separate trees are constructed for words in the main and filler lexicons. Furthermore, several copies may be instantiated for the two, depending on the -Nlextree configuration argument.

The lexical tree representation is motivated by the fact that Most active HMMs are word-initial models, decaying rapidly subsequently. e.g. On 60K-word Hub-4 task, 55 word-initial. But, no. of distinct word-initial model types are much fewer:

START S-T-AA-R-TD
STARTING S-T-AA-R-DX-IX-NG
STARTED S-T-AA-R-DX-IX-DD
STARTUP S-T-AA-R-T-AX-PD
START-UP S-T-AA-R-T-AX-PD

Therefore, it makes sense to use so called the prefix-tree to represent the lexicon and maximizing sharing among words

[Under construction] [Insert p.21 of Ravi’s slides at here]

Triphone representation

[Under construction] [Insert p.22 and p.23 of Ravi’s slides at here]

Integration with language model

[Under construction] [Insert p.24-32 of Ravi’s slides at here]

LM lookahead

One of the key of fast search is to make sure high level information can be used as soon as possible. Here comes the idea of
Unigram LM probability is factored into the tree using technique [Need to create. described in Spoken Language Dialogue]

10.6.3 Language model

The main language model (LM) used by the Sphinx decoder is a conventional bigram or trigram backoff language model. The CMU-Cambridge SLM toolkit is capable of generating such a model from LM training data. Its output is an ascii text file. But a large text LM file can be very slow to load into memory. To speed up this process, the LM must be compiled into a binary form. The code to convert from an ascii text file to the binary format is available at SourceForge in the CVS tree, in a module named share.

Unigrams, Bigrams, Trigrams, LM Vocabulary

A trigram LM primarily consists of the following:

- **Unigrams**: The entire set of words in this LM, and their individual probabilities of occurrence in the language. The unigrams must include the special beginning-of-sentence and end-of-sentence tokens: `<s>`, and `</s>` respectively.

- **Bigrams**: A bigram is mathematically $P(\text{word2} \mid \text{word1})$. That is, the conditional probability that word2 immediately follows word1 in the language. An LM typically contains this information for some subset of the possible word pairs. That is, not all possible word1 word2 pairs need be covered by the bigrams.

- **Trigrams**: Similar to a bigram, a trigram is $P(\text{word3} \mid \text{word1}, \text{word2})$, or the conditional probability that word3 immediately follows a word1 word2 sequence in the language. Not all possible 3-word combinations need be covered by the trigrams.

The vocabulary of the LM is the set of words covered by the unigrams.

The LM probability of an entire sentence is the product of the individual word probabilities. For example, the LM probability of the sentence "HOW ARE YOU" is:

\[
P(\text{HOW} \mid <s>) * P(\text{ARE} \mid <s>, \text{HOW}) *
\]
**Pronunciation and Case Considerations**

In Sphinx, the LM cannot distinguish between different pronunciations of the same word. For example, even though the lexicon might contain two different pronunciation entries for the word READ (present and past tense forms), the language model cannot distinguish between the two. Both pronunciations would inherit the same probability from the language model.

Secondly, the LM is case-insensitive. For example, it cannot contain two different tokens READ and read.

The reasons for the above restrictions are historical. Precise pronunciation and case information has rarely been present in LM training data. It would certainly be desirable to do away with the restrictions at some time in the future.

**Binary LM File**

The binary LM file (also referred to as the LM dump file) is more or less a disk image of the LM data structure constructed in memory. This data structure was originally designed during the Sphinx-II days, when efficient memory usage was the focus. In Sphinx-3, however, memory usage is no longer an issue since the binary file enables the decoder to use a disk-based LM strategy. That is, the LM binary file is no longer read entirely into memory. Rather, the portions required during decoding are read in on demand, and cached. For large vocabulary recognition, the memory resident portion is typically about 10-20 the bigrams, and 5-10.

Since the decoder uses a disk-based LM, it is necessary to have efficient access to the binary LM file. Thus, network access to an LM file at a remote location is not recommended. It is desirable to have the LM file be resident on the local machine.

The binary dump file can be created from the ascii form using the lm3g2dmp utility, which is part of the Sphinx-II distribution, and also available as standalone code, as mentioned before. (The header of the dump file itself contains a brief description of the file format.)
Silence and Filler Words

Language models typically do not cover acoustically significant events such as silence, breath-noise, UM or UH sounds made by a person hunting for the right phrase, etc. These are known generally as filler words, and are excluded from the LM vocabulary. The reason is that a language model training corpus, which is simply a lot of text, usually does not include such information.

Since the main trigram LM ignores silence and filler words, their "language model probability" has to be specified in a separate file, called the filler penalty file. The format of this file is very straightforward; each line contains one word and its probability, as in the following example:

```
++UH++ 0.10792
++UM++ 0.00866
++BREATH++ 0.00147
```

The filler penalty file is not required. If it is present, it does not have to contain entries for every filler word. The decoder allows a default value to be specified for filler word probabilities (through the `-fillprob` configuration argument), and a default silence word probability (through the `-silprob` argument).

Like the main trigram LM, filler and silence word probabilities are obtained from appropriate training data. However, training them is considerably easier since they are merely unigram probabilities.

Filler words are invisible or transparent to the trigram language model. For example, the LM probability of the sentence "HAVE CAR <sil> WILL TRAVEL" is:

```
P(HAVE — <s>) *
P(CAR — <s>, HAVE) *
P(<sil>) *
P(WILL — HAVE, CAR) *
P(TRAVEL — CAR, WILL) *
P(</s> — WILL, TRAVEL)
```

Language Weight and Word Insertion Penalty

During recognition the decoder combines both acoustic likelihoods and language model probabilities into a single score in order to compare vari-
ous hypotheses. This combination of the two is not just a straightforward product. In order to obtain optimal recognition accuracy, it is usually necessary to exponentiate the language model probability using a language weight before combining the result with the acoustic likelihood. (Since likelihood computations are actually carried out in the log-domain in the Sphinx decoder, the LM weight becomes a multiplicative factor applied to LM log-probabilities.)

The language weight parameter is typically obtained through trial and error. In the case of Sphinx, the optimum value for this parameter has usually ranged between 6 and 13, depending on the task at hand.

Similarly, though with lesser impact, it has also been found useful to include a word insertion penalty parameter which is a fixed penalty for each new word hypothesized by the decoder. It is effectively another multiplicative factor in the language model probability computation (before the application of the language weight). This parameter has usually ranged between 0.2 and 0.7, depending on the task.

### 10.6.4 Pruning

Each entry in the control file, or utterance, is processed using the given input models, and using the Viterbi search algorithm. In order to constrain the active search space to computationally manageable limits, pruning is employed, which means that the less promising hypotheses are continually discarded during the recognition process. There are two kinds of pruning in s3.X, beam pruning and absolute pruning.

#### Viterbi Pruning or Beam Pruning

Each utterance is processed in a time-synchronous manner, one frame at a time. At each frame the decoder has a number of currently active HMMs to match with the next frame of input speech. But it first discards or deactivates those whose state likelihoods are below some threshold, relative to the best HMM state likelihood at that time. The threshold value is obtained by multiplying the best state likelihood by a fixed beamwidth. The beamwidth is a value between 0 and 1, the former permitting all HMMs to survive, and the latter permitting only the best scoring HMMs to survive.

Similar beam pruning is also used in a number of other situations in the decoder, e.g., to determine the candidate words recognized at any time, or to determine the component densities in a mixture Gaussian that are
closest to a given speech feature vector. The various beamwidths have to be determined empirically and are set using configuration arguments.

**Histogram Pruning or Absolute Pruning**

Absolute Pruning. Even with beam pruning, the number of active entities can sometimes become computationally overwhelming. If there are a large number of HMMs that fall within the pruning threshold, the decoder will keep all of them active. However, when the number of active HMMs grows beyond certain limits, the chances of detecting the correct word among the many candidates are considerably reduced. Such situations can occur, for example, if the input speech is noisy or quite mismatched to the acoustic models. In such cases, there is no point in allowing the active search space to grow to arbitrary extents. It can be contained using pruning parameters that limit the absolute number of active entities at any instant. These parameters are also determined empirically, and set using configuration arguments.

- **-beam**: Determines which HMMs remain active at any given point (frame) during recognition. (Based on the best state score within each HMM.)

- **-pbeam**: Determines which active HMM can transition to its successor in the lexical tree at any point. (Based on the exit state score of the source HMM.)

- **-wbeam**: Determines which words are recognized at any frame during decoding. (Based on the exit state scores of leaf HMMs in the lexical trees.)

- **-maxhmmpf**: Determines the number of HMMs (approx.) that can remain active at any frame.

- **-maxwpf**: Controls the number of distinct words recognized at any given frame.

- **-maxhistpf**: Controls the number of distinct word histories recorded in the backpointer table at any given frame.

- **-subvqbeam**: For each senone and its underlying acoustic model, determines its active mixture components at any frame.

In order to determine the pruning parameter values empirically, it is first necessary to obtain a test set, i.e., a collection of test sentences not
used in any training data. The test set should be sufficiently large to ensure statistically reliable results. For example, a large-vocabulary task might require a test set that includes a half-hour of speech, or more.

It is difficult to tune a handful of parameters simultaneously, especially when the input models are completely new. The following steps may be followed to deal with this complex problem.

1. To begin with, set the absolute pruning parameters to large values, making them essentially ineffective. Set both `-beam` and `-pbeam` to 1e-60, and `-wbeam` to 1e-30. Set `-subvqbeam` to a small value (e.g., the same as `-beam`). Run the decoder on the chosen test set and obtain accuracy results. (Use default values for the LM related parameters when tuning the pruning parameters for the first time.)

2. Repeat the decoder runs, varying `-beam` up and down, until the setting for best accuracy is identified. (Keep `-pbeam` the same as `-beam` every time.)

3. Now vary `-wbeam` up and down and identify its best possible setting (keeping `-beam` and `-pbeam` fixed at their most recently obtained value).

4. Repeat the above two steps, alternately optimizing `-beam` and `-wbeam`, until convergence. Note that during these iterations `-pbeam` should always be the same as `-beam`. (This step can be omitted if the accuracy attained after the first iteration is acceptable.)

5. Gradually increase `-subvqbeam` (i.e., towards 1.0 for a narrower setting), stopping when recognition accuracy begins to drop noticeably. Values near the default are reasonable. (This step is needed only if a sub-vector quantized model is available for speeding up acoustic model evaluation.)

6. Now gradually increase `-pbeam` (i.e., towards 1.0), stopping when recognition accuracy begins to drop noticeably. (This step is optional; it mainly optimizes the computational effort a little more.)

7. Reduce `-maxhmmpf` gradually until accuracy begins to be affected. Repeat the process with `-maxwpf`, and then with `-maxhistpf`. (However, in some situations, especially when the vocabulary size is small, it may not be necessary to tune these absolute pruning parameters.)

In practice, it may not always be possible to follow the above steps strictly. For example, considerations of computational cost might dictate that the absolute pruning parameters or the parameters of the GMM
computation first. Let’s assume if we use Sub-vector quantization as the
GMM computation technique. The following is a practical way to tune the
parameters.  

10.6.5 Phoneme look-ahead

[Add Jahanzeb’s idea of phoneme lookahead at here.]

10.7 Architecture of GMM Computation in decode

The technique described in this section can be found in “Four-Level Cate-
gorization Scheme of Fast GMM Computation Techniques in Large Vocab-
ularv Continuous Speech Recognition Systems” by Arthur Chan. One can
treat the computation of GMM as a 4-level process.

- Frame-level
- GMM-level
- Gaussian-level
- Component-level

Approximation can be done at every level such that accuracy will not
be sacrificed. Several techniques were implemented in Sphinx

10.7.1 Frame-level optimization

Algorithms that decide whether a frame’s GMM scores should be computed
or skipped are categorized as frame-layer algorithms. In our discussion, we
will assume the score of a skipped frame will be copied from the most re-
cently computed frame.  

420040927(Arthur): This section is copied from Ravi’s codewalk. It will later be ex-
panded such that -ci.pbeam , -ds will also be employed and used for tuning.

5This implementation can preserve transition information.
only every other frame, which we call this simple down-sampling (SDS). One can apply this technique in Sphinx 3.X, by specifying `-ds N`.

Other than SDS, we also evaluate another scheme in which a VQ codebook is trained from all means of GMMs of a set of trained acoustic models. Then, in decoding, every frame’s feature vector is quantized using that codebook. A frame is skipped only if its feature vector was quantized to a codeword which is the same as that of the previous frame. We call this method VQ-based Down-Sampling (VQDS). This can be used by `-cond_ds`, if this flag is specified, flag `-gs` is necessary to be applied.

Comparatively speaking, SDS provide more gain in speed but cause some degradation and it is very ideal for user who seek for speed.

### 10.7.2 GMM-level optimization

Algorithms that ignore some GMMs in the computation in each computed frame are assigned to the GMM-layer. One representative technique is context-independent (CI) GMM-based GMM selection (CIGMMS) experimented by Lee et al. in Julius. Briefly the algorithm can be implemented as follows. At every frame, CI GMMs’ scores are first computed and a beam is applied to these scores. For all context-dependent (CD) GMMs, if the corresponding CI GMM’s score is within the beam, compute the detail CD GMM’s score. If not, the CD GMM’s score is backed-off by the corresponding CI GMM’s score.

This scheme is highly effective in reducing GMM computation. However, because some scores are backed-off, they become the same when they are fed into the search module. As a consequence, beam pruning becomes less effective.

This technique can be used by specified `-ci_pbeam`. The range can be specified is between $10^{-5}$ to $10^{-80}$.

### 10.7.3 Gaussian-level optimization: VQ-based Gaussian Selection and SVQ-based Gaussian Selection

Usually only a few Gaussians will dominate the likelihood of a GMM and different techniques are used to decide which Gaussian dominates the likelihood computation. We categorize these techniques as part of the Gaussian layer. Generally a rough model, either vector-quantizer (VQ)
The major issue in using any Gaussian selection techniques is the need to trade-off between rough model computation (e.g., the VQ codebook) and accuracy degradation. Usually, a more detailed model (e.g., a higher-ordered VQ codebook) gives better accuracy, but results in more computation.

Usually, the neighborhood for a particular code word is computed by thresholding the distance of Gaussian mean from the codeword. (cite Bochierri's paper). The consequence of using this scheme is that there will be cases which a neighborhood contains no Gaussian at all. Hence, in work by Gales et al., it is suggested that one could fix this problem by setting a minimum number of Gaussian in a certain neighborhood. Note that in work by Douglas (cite in Douglas's work) suggested that instead of using the code word as a reference point of deciding the neighborhood, one could use the closest Gaussian's mean as the reference point. In this case, one can always use the closest point as one of the Gaussian in the neighborhood.

One can use -gsfn to specify the gaussian selector map. The Gaussian selector map can be generated by gselect.

In Sphinx 3.X, we focused on two techniques which made use of simple VQ as a Gaussian selector (VQGS) and SVQ as a Gaussian selector (SVQGS). We ignored tree-based techniques because of the practical difficulty in then applying adaptation techniques such as Maximum Likelihood Linear Regression (MLLR). One can use -svqfn to specify sub-VQ map. The sub-vector quantization can be generated by gausubvq.

### 10.7.4 Gaussian/Component-level optimization: Sub-vector quantization

As in the full-feature space, the distribution of features in a projection of the full-space (or subspace) can be approximated as a GMM. The full-space likelihood can thus be obtained by summing individual sub-spaces likelihood. Similar to situation in the full-space, only few Gaussians will dominate the subspace likelihood in a particular subspace. Therefore, techniques can be used to choose the best Gaussians in individual sub-spaces and combined them. We categorize algorithms which make use of this fact to be component-layer algorithm. We will focus one representative technique in this layer, sub-vector quantization (SVQ) in which VQ
was used as a Gaussian selector in subspaces.

The sub-vector quantization can be generated by `gausubvq`.

```
$ gausubvq
  -mean ./hub4_cd_continuous_8gau_1s_c_d_dd/means
  -var ./hub4_cd_continuous_8gau_1s_c_d_dd/variances
  -mixw ./hub4_cd_continuous_8gau_1s_c_d_dd/mixture_weights
  -svspec 0-38
  -iter 20
  -svqrows 16
  -subvq svq.out
```

in a file called svq.out. This can be used in `decode`

### 10.7.5 Interaction between different level of optimization

Our categorization scheme results in a layered architecture for GMM computation and many fast GMM techniques can be categorized into one of the layers\(^6\). The advantage of this scheme is that one can follow a conceptual model when implementing different algorithms. It also simplifies the studies of interaction of different schemes. For example, once we understand that two techniques are in the same-layer (such as VQGS and SVQGS), we will probably not want to implement them in the same system, as that can only result in higher overhead.

### 10.7.6 Related tools, `gs_select`, `gs_view`, `gausubvq`

[To be completed]

---

\(^6\)We also note that some techniques has the characteristics of multiple layers.
10.7.7 Interaction between GMM computation and search routines in Sphinx 3.X

The performance of fast GMM computation techniques is usually less effective when a tighter beam is used in search. From the perspective of our conceptual model, pruning can be regarded as a GMM-layer algorithm, and as such, only affects the layers above, namely, the GMM and frame layers.

We summarize the effect below.

Relationship with Frame-Layer: We assume skipped frames’ scores are copied from previous frames’ scores. However, the search module will decide whether a clustered-GMM is active or not based on pruning. There will be problematic situations where some GMMs are active in the current frame but deactivated in previous frame. Hence, recomputation of active states is necessary. When the beam is tight, recomputation can be time-consuming and can cancel out the computation gain obtained from downsampling.

7 One can also deactivate the state. However, in our preliminary experiment, we found that deactivation can result in no valid paths at the final frame.

Relationship with GMM-Layer: As the GMM’s scores will feed into the search module, one observation is that if CIGMMS applied, the search time increases. If the CI scores’ beam (mentioned in Section ??) is tight, CIGMMS will cause many CD-GMMs’ scores to be the same and also narrows the range of scores. Hence, the search module will be more computationally intensive. In practice, this problem can be solved by tightening the Viterbi beam.

10.8 Overall search structure of decode

In the last two sections, we described the search and the GMM computation. The following algorithm provides a unified point of view of what’s going on.

Initialize start state of $<s>$ with path-score $= 1$;

[To be completed]

find final $</s>$ BP table entry and back-trace through table to retrieve result;
10.9 Multi-pass systems using Sphinx 3.X

10.9.1 Word Lattice

During recognition the decoder maintains not just the single best hypothesis, but also a number of alternatives or candidates. For example, REED is a perfectly reasonable alternative to READ. The alternatives are useful in many ways: for instance, in N-best list generation. To facilitate such post-processing, the decoder can optionally produce a word lattice output for each input utterance. This output records all the candidate words recognized by the decoder at any point in time, and their main attributes such as time segmentation and acoustic likelihood scores.

The term "lattice" is used somewhat loosely. The word-lattice is really a directed acyclic graph or DAG. Each node of the DAG denotes a word instance that begins at a particular frame within the utterance. That is, it is a unique word,start-time pair. (However, there could be a number of end-times for this word instance. One of the features of a time-synchronous Viterbi search using beam pruning is that word candidates hypothesized by the decoder have a well-defined start-time, but a fuzzy range of end-times. This is because the start-time is primarily determined by Viterbi pruning, while the possible end-times are determined by beam pruning.)

There is a directed edge between two nodes in the DAG if the start-time of the destination node immediately follows one of the end times of the source node. That is, the two nodes can be adjacent in time. Thus, the edge determines one possible segmentation for the source node: beginning at the source’s start-time and ending one frame before the destination’s start-time. The edge also contains an acoustic likelihood for this particular segmentation of the source node.

Note: The beginning and end of sentence tokens, <s> and </s>, are not decoded as part of an utterance by the s3.X decoder. However, they have to be included in the word lattice file, for compatibility with the older Sphinx-3 decoder software. They are assigned 1-frame segmentations, with log-likelihood scores of 0. To accommodate them, the segmentations of adjacent nodes have to be “fudged” by 1 frame.

Word Lattice File Format

A word lattice file essentially contains the above information regarding the nodes and edges in the DAG. It is structured in several sections, as follows:
1. A comment section, listing important configuration arguments as comments
2. Frames section, specifying the number of frames in utterance
3. Nodes section, listing the nodes in the DAG
4. Initial and Final nodes (for <s> and </s>, respectively)
5. BestSegAscr section, a historical remnant now essentially empty
6. Edges section, listing the edges in the DAG

The file is formatted as follows. Note that any line in the file that begins with the # character in the first column is considered to be a comment.

# getcwd: <current-working-directory>
# -logbase <logbase-in-effect>
# -dict <main lexicon>
# -fdict <filler lexicon>
# ... (other arguments, written out as comment lines)
#
Frames <number-of-frames-in-utterance>
#
Nodes <number-of-nodes-in-DAG> (NODEID WORD STARTFRAME FIRST-ENDFRAME LAST-ENDFRAME)

<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>
<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>
<Node-ID> <Word-String> <Start-Time> <Earliest-End-time> <Latest-End-Time>
...

  ... (for all nodes in DAG)
#
# Initial <Initial-Node-ID>
Final <Final-Node-ID>
#
BestSegAscr 0 (NODEID ENDFRAME ASCORE)
#
Edges (FROM-NODEID TO-NODEID ASCORE)

<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>
<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>
<Source-Node-ID> <Destination-Node-ID> <Acoustic Score>
... (for all edges in DAG) End

Note that the node-ID values for DAG nodes are assigned sequentially, starting from 0. Furthermore, they are sorted in descending order of their earliest-end-time attribute.

10.9.2 astar

Given a lattice, the program astar is able to generate the N-best results.

In the following example, a file name abc.lat is generated in a directory ./an4/. The following command will rescore generate the N-Best list for each of these lattice.

```
$ src/programs/astar
-mdeffn ./hub4_cd_continuous_8gau_1s_c_d_dd/hub4opensrc.6000.mdef
-fdict ./filler.dict
-dict ./an4.dict
-lm ./an4.ug.lm.DMP
-langwt 13.0
-inspen 0.2
-ctlfn ./an4.ctl
-inlatdir ./an4/
-logbase 1.0003
-backtrace 1
```

10.9.3 dag

Given a lattice, the program dag is able to get the best path through the lattice.

```
$ src/programs/dag
-mdef hub4_cd_continuous_8gau_1s_c_d_dd/hub4opensrc.6000.mdef
```
-fdict filler.dict
-dict an4.dict
-lm an4.ug.lm.DMP
-lw 13.0
-wip 0.2
-ctl an4.ctl
-inlatdir an4/
-logbase 1.0003
-backtrace

The usage of **dag** is very similar to astar. Please consult the manual for **astar**.

### 10.10 Other tools inside S3.X packages

#### 10.10.1 align

This tool can create phone-level alignment given the transcription. The tool will guess the timing information for each word in the transcription.

For example:

```
$ align
-logbase 1.0003
-mdef ./hub4opensrc.6000.mdef
-mean ./means
-var ./variances
-mixw ./mixture_weights
-tmat ./transition_matrices
-feat 1s_c_d_dd
-topn 1000
-beam 1e-80
-senmgaufn .cont.
-fdict .//model/lm/an4/filler.dict
```
10.10.2 allphone

This tool can do generate the best phoneme sequence that matches a wave forms.

Example.

$ allphone
    -logbase 1.0003
    -mdeffn ./hub4opensrc.6000.mdef
    -meanfn ./means
    -varfn ./variances
    -mixwfn ./mixture_weights
    -tmatfn ./transition_matrices
    -feat 1s_c_d_dd
    -topn 1000
    -beam 1e-80
    -senmgaufn .cont.
    -ctlfn ./an4.ctl
    -cepdir ./an4/
    -phsegdir ./

10.11 Using the sphinx-iii decoder with semi-continuous and continuous models

There are two flags which are specific to the type of model being used, the rest of the flags are independent of model type. The flags you need to change to switch from continuous models to semi-continuous ones are:

- the -senmgaufn flag would change from ".cont." to ".semi."
- the -feat flag would change from the feature you are using with continuous models to the feature you are using with the semicontinuous models (usually it is s3.1x39 for continuous models and s2.4x for semi-continuous models)

Some of the other decoder flags and their usual settings are as follows:

- logbase 1.0001
- bestpath 0
- mdef $mdef
- senmgau .cont.
- mean ACMODDIR/means
- var ACMODDIR/variances
- mixw ACMODDIR/mixture_weights
- tmat ACMODDIR/transition_matrices
- langwt 10.5
- feat s3.1x39
- topn 32
- beam 1e-80
- nwbeam 1e-40
- dict dict
- fdict fdict
- fillpen fillpen
- lm lmfie
- inspen 0.2
- ctl ctl

219
-ctloffset ctloffset
-ctlcoun tctlcount
-cepdir cepdir
-bptblsize 400000
-matchseg matchfile
-outlatdir outlatdir
-agc none
-varnorm yes
Chapter 11

Speaker Adaptation using Sphinx

11.1 Speaker Adaptation

Author: David Huggins-Daines, Arthur Chan, Editor: Arthur Chan

In the previous chapters, various tools for creating acoustic model and language model for a speech application were introduced. One problem of the maximum likelihood framework used in Sphinx is that if there are mismatch between the training and testing conditions, the recognition rate will be significantly affected.

There are several sources of mismatch:

1. **Environmental Mismatches** The influence of additive and convolutive noise can significantly change the characteristics of the acoustic models.

2. **Speaker Variabilities** Inter-speaker variabilities of speech, e.g. Speaker A and Speaker B will utter the same word with very different acoustic characteristics.

3. **Language Mismatches** The difference of usage in the training and testing corpora can reduce the power of the language model in decoding.

In this chapter, we will discuss techniques of speaker adaptation provided by Sphinx. This chapter currently discusses only acoustic model adaptation, which can alleviate the first two problems mentioned above, but not the third.
11.2 Different Principles of Speaker Adaptation

11.2.1 In terms of the mode of collecting adaptation data

Generally, speaker adaptation assumes that an acoustic model already exists and a new set of adaptation data is available. This new set of adaptation can be collected by asking the users to speak following a predefined transcription. This is called *supervised adaptation*. The process of collecting data for adaptation is usually called “enrollment”.

The process of enrollment can be tedious and time-consuming and many users dislike it. In some applications, it is also impossible to allow such a design. For example, for telephony applications, users may be travelling while they make the calls. Asking the users to provide enrollment data can be very annoying.

However, supervised adaptation has its advantages. The performance of the adapted system is usually better because the transcription is known. Therefore, a lot of systems such as dictation usually ask for users to give 5 minutes to 30 minutes of enrollment data (sometimes more). This usually gives much better results.

As opposed to supervised adaptation, *unsupervised adaptation* attempts to determine the transcription by using an automatic speech recognizer. This avoids the need for enrollment. However, the performance of the resulting adaptation will usually be poorer because it is inevitable that the speech recognizer will make some mistakes in determining the transcription.

So which way to go? The choice of using supervised and unsupervised adaptation depends on many human factors. Major one is whether the user is patient enough to give enrollment data. In terms of performance, it can be very hard to foresee how much gain one can get from speaker adaptation. Usually, the number is relatively 5-15unsupervised adaptation, the accuracy can even end up worse than the original system. Therefore, the use of each of these modes of adaptation should be a subject of serious consideration.

11.2.2 In terms of technique of parameter estimation

There are two major ways to do speaker adaptations. One is the so called “maximum a-posterior” (MAP) re-estimation of the parameters. One is maximum likelihood linear regression (MLLR).
We will mainly discuss speaker adaptation using MLLR. The use of regression analysis to find the best linear (or non-linear) relation between two sets of data is well-known. Linear regression consists of trying to find the terms $A$ and $b$ in the following set of equations:

$$y = Ax + b$$

The standard way of solving this problem is using the least minimum square estimate (LMSE) method. Maximum likelihood linear regression is basically putting linear regression into the Baum-Welch estimation framework. The consequence of this technique is that the transform that is estimated is optimal according to the maximum likelihood criterion.  

The use of MLLR can allow speaker adaptation of all phonemes even if there is only few utterances of adaptation data. In such a case, all the data will be used to estimate a single transformation matrix. This also adds some robustness to incorrect transcriptions, making MLLR suitable for unsupervised adaptation even with fairly noisy data.

The problem of single-transformation MLLR is that each phone usually has different acoustic characteristic and the transformations are usually different. Therefore, in order to make more efficient use of the adaptation data, it is necessary to allow different phones to have their own speaker adaptation matrix, or to tie data from different phones together and create the same transformation matrix for them. Usually, this technique is called multiple regression classes.

In practice, how many regression classes should be used is usually determined by validation using testing data.

MAP adaptation is really a variant of the acoustic model training procedure. The standard Baum-Welch algorithm results in a maximum likelihood estimate of the HMM model parameters given the training data and an initial model, which serves only to fill in the “hidden” data in the re-estimation formula (i.e. the state sequence and observation counts). The MAP technique extends this to produce a maximum a-posteriori estimate, where re-estimation takes into account the prior distribution of the model parameters, which in the case of speaker adaptation is (or can be directly calculated from) the parameters of the baseline, speaker-independent model.

In the case where only the means are adapted (which is sufficient for continuous density HMMs), this can be simplified to an interpolation be-

---

1The term “regression” means “going back to the linear relation” when it first used by Fisher. (Is that true?)
tween the baseline acoustic mean and the ML estimate of the adaptation data, weighted by the observation count of the adaptation data and the variance of the baseline and adaptation data models. Intuitively, we move the parameters for each senone observed in the adaptation data such that we maximize the likelihood of the adaptation data, while at the same time minimizing the variance.

### 11.3 MLLR with SphinxTrain

This section describes the use of the procedure of using speaker adaptation facilities of Sphinx. Sphinx provides facilities for supervised adaptation.

1. Given a previous acoustic model and new set of adaptation data. bw (refer to Chapter ?) is first used to collect the sufficient statistics count. For MLLR, what we need is just to have the mean vector. For MAP, we need the mean and variance.

2. Estimate the transformation matrix using the program mllr_solve.

3. Apply the transformation to the mean vector

#### 11.3.1 Using bw for adaptation

As it is described in Chapter, bw is the the program which carries out forward-backward algorithm and collects the sufficient statistics. MLLR can be thought as a regression matrix estimator operated on these sufficient statistics. The actual command is:

```bash
$ bw
    -moddeffn mdef
    -mixwfn mixw
    -meanfn mean
    -varfn var
    -tmatfn tmat
```

---

2You will find many people called it posterior probabilities. They are correct. But to learn the reason, I need to explain the mathematics so I tried to avoid it. :-}
-dictfn dict
-fdictfn filler_dict
-cepdir feat_dir
-cepext .mfc
-lsnfn transcripts_fn
-meanreest yes
-varreest yes
-2passvar yes
-feat 1s_c_d_dd
-ceplen 13
-ctlfn adapt.ctl
-accumdir accumdir
-agc none
-cmn current

The command will use the adaptation data found in adapt.ctl to gather the mean and variance statistics in the directory accumdir.

11.3.2 mllr_solve

mllr_solve is the program that actually computes the transformation matrix.

$ mllr_solve
   -outmllrfn adapt.matrix
   -accumdir accumdir
   -meanfn mean
   -varfn vars
   -moddefn mdef
   -cdonly yes

The command will make use of the mean accumulators in director accumdir and estimate the vector. You also see the mean and variances are loaded in. The MLLR algorithm is basically an iterative algorithm so theoretically if you do the above process for more turns, the matrix you get will give better performance. In practice, 1-2 iterations are usually enough.
Notice that we only apply the transformation on the CD-senones as we specified -cdonly at here. Notice that when you specify option -cdonly, you also need to specify a model definition because it provides the mapping to determine which model is CI or CD.

In some cases, e.g. when you are using CI-based GMM selection, you may consider to adapt also CI-senones. Notice that, though, the effect of transformation-based adaptation (like MLLR) on the fast search is sometimes less studied. We have found that in general, it does not hurt to adapt the CI senones as well, so it is reasonable to omit the -cdonly option.

11.3.3 Using mllr_transform to do offline mean transformation

There are two ways to use the resulting matrix, one is to use it to transform the mean vectors offline, which is shown in this subsection. One is to use it to transform mean vectors on-line which we will show in next subsection.

$ mllr\_transform
 -\text{inmeanfn} \text{ mean}
 -\text{outmeanfn} \text{ out.mean}
 -\text{mllrmatrix} \text{ adapt.matrix}
 -\text{cdonly} \text{ yes}

In such a case a new set of means, out.mean will be formed and you can just use them in speech recognition.

11.3.4 On-line adaptation using Sphinx 3.0 and Sphinx 3.X decoder

In Sphinx 3.0 and Sphinx 3.X decoders (decode\_anytopo and decode), one can apply the adaptation matrix on-line by specifying a MLLR control file. For example, if you want the n-sentence to use regression matrix \text{abc} to adapt the mean on-line. Then you can create a file call \text{ctl.mllr} that contains

$ \text{abc}

as the n-th line of the file. You can then apply this transformation by specifying -ctl\_mllr in the command line of decode. Then when the
recognizer is decoding the n-th sentence, transformation matrix abc will be used to transform the mean.

In addition, Sphinx 3.6 will allow you to specify a default adaptation matrix, using the -mllr option to specify the transformation matrix file.

### 11.4 MAP with SphinxTrain

The process of doing MAP adaptation is similar to that of MLLR. First, you must collect statistics from the adaptation data using the bw tool, as described above. It is important that you specify the -2passvar yes option to bw, because accurate estimation of the variances is crucial to MAP adaptation, and because otherwise, the variance may be negative (!) for unobserved mixtures. Next, you must use norm, as in training, to create a maximum-likelihood estimate of the mean and variance parameters for the adaptation data:

```
$ norm -accumdir accumdir -meanfn means.ml -varfn variances.ml -dcountfn dcount.ml -feat 1s_c_d_dd
```

These estimated means and variances are then interpolated with the baseline model, using the map_adapt tool, to produce a MAP estimate of the means:

```
$ map_adapt -mapmeanfn map_means -simeanfn means -sivarfn variances -mlmeanfn means.ml -mlvarfn variances.ml -mlcntfn dcount.ml
```

There are two drawbacks and one great advantage to using MAP. First, the bad news: MAP is not suitable for unsupervised adaptation. It is very important that the transcriptions of the adaptation data be correct; you may wish to consider force-aligning them, even. A somewhat lesser concern is that since MAP requires re-estimation of the means, there is no "on-line" adaptation and thus it is necessary to store a separate copy of the mean parameter file for each speaker.

The good news is that MAP can make much more effective use of large amounts of adaptation data. If you have more than 50 sentences of adaptation data, it is a good idea to use MAP in addition to MLLR. To do this, the recommended procedure is to perform MLLR adaptation as described above, using mllr_transform to generate a new means file, then re-run bw on the adaptation data using the transformed means, and finally do MAP adaptation as described here.
Chapter 12

Development using Sphinx

Author: Yitao Sun and Arthur Chan, Editor: Arthur Chan

12.1 Sphinx’s recognizer Application Programming Interface (API)

In previous chapters, the property of a speech recognizer was described. It is important to realize that that is mainly for the core speech recognition engine. Using the speech recognizer in real-life application has other considerations:

1. **Resource allocation** Resource may be required to be change dynamically as a per utterance perspective.

2. **Real-time response** The system is required to provide quick and natural response.

3. **Programming Interface** The developer can use the functionalities of the recognizer through a set of application programmer interface (API).

This section discuss these issues and provide a brief introduction of the programming using Sphinx. We will describe each issue in detail.
12.2 Issues in live-mode recognition

We will describe three important issues in using the live mode decoder, that include how to segment the speech, the impact of recognizer’s speed and the consideration of delay.

12.2.1 Issues in speech segmentation

One of the major difference between the batch mode recognizer and the live mode recognizer is that a clear begin and end time of the whole utterance is usually well-defined before the decoding. While, the live mode decoder may have no knowledge about the length of the whole utterances. For example, in so-called continuous recording mode of the recognizer. The audio data was continuously captured and feed into the the live mode recognizer. To avoid delay, usually the application developer will define a fix and usually short speech segment’s size when inputting the speech signal into the recognizer.

Another way to decide which segment is correct is to parse the sentence with and end-pointer. The segmented output will be put into the recognizer. Internally, sphinx used an energy-histogram-based end-pointer to decide an end-pointer. This is usually not the optimal way to do end-pointing. Ziad Al Bawab and Arthur Chan have created a version of Sphinx 3.X that incorporated and end-pointer, ep. You can find this information at,

www.cs.cmu.edu/~archan/share/s3ep.tgz

12.2.2 Issues in speed and accuracy performance

Usually, real-life application require fast response from the speech recognizer. This requires so called real-time processing from the speech recognizer. As the design of Sphinx give the programmer the freedom to use different configuration of acoustic models and tune the system themselves. It also become the users or developers responsibilities to make sure the recognizer can be ran in real-time or faster than real-time.

One useful measure to allow us to have some feeling of the speed of the code is the xRT scale. For example, for a 1s waveform, if the processing
requires $N$ to process, then we call it the system as \( N \times \text{RT} \) system. Obviously, to have real-time response, $N$ has to be a number smaller than 1 or else the response of the recognizer would be slower than its input.

The first important consideration of building a real-time speech recognition system is to decide whether it is feasible to build such a system. In 2004, it is possible to tune live_pretend (using techniques described in Chapter 10) to be as fast as smaller than \( 1 \times \text{RT} \) for a task which has 20k words with perplexity around 80 using trigrams language model. The system will still give very reasonable performance compared to the un-tuned system. (within 5% relative degradation)

However, it is not always possible to make the recognizer less than \( 1 \times \text{RT} \) with reasonable performance due to limitation of computation power. For example, in 2004, broadcast news (65k words) is one task that a 1-pass speech recognizer such as Sphinx will have difficulty to give reasonable performance in \( 1 \times \text{RT} \). For example, in some task, 10 fold speed-up would mean 20 determined by experiments.

It is important to realize that it is always possible to the recognizer down to a certain speed, this will, however, always cause degradation of accuracy of the recognizer.

As another sidenote, usually speech recognition will not be the only component of a dialogue system. Other components such as end-pointer, speech synthesis can also require amount of computation. Therefore, it is important to profile individual components in troubleshooting your system.

### 12.2.3 Issues in delay in response

Delay of a speech recognizer can be caused by several reasons. Obviously, if the speed of the recognizer cannot catch up with the audio capturing routine, it will cause a substantial delay. This sub-section assumes the users of Sphinx have already tuned the system under \( 1 \times \text{RT} \).

The most major factor that affect the delay of speech recognition system is the mechanism how the audio-capturing routines and the recognizer interact with each other. For example, if the audio routine only send data to recognizer every 10s, the system will bound to have long delay.

Internally, sphinx use a set of reasonably portable libraries for audio capturing. However, by its own design, capturing the audio data from the audio device will become a blocking call. This can also cause unreasonable delay. Therefore, these routines were not exposed to the public.
We, therefore, recommend the user take care of the problem of audio capturing separate from using the recognizer. For PC hardware, these problems can be solved by using libraries such as portaudio.

www.portaudio.org

Portaudio provides a set of mechanism to reduce the delay of the system, please consult its documentation to learn the howto.

12.3 Using the live mode recognizer

There are several major important process in using Sphinx’s live mode recognizer.

1. **Resource Allocation/Deallocation** In a recognizer life-time, several important resources need to load in such as dictionary and acoustic model. This initialization/uninitialization from the command-line front-end is done by calling ld_init/ld_finish.

2. **Decoding Session Begin/End** [Need to rewrite] For every utterance, Sphinx start recognition by starting ld_begin_utt and end that by ld_end_utt. At every utterance, it is possible to set dynamic resource such as language models and transformation matrices for adaptation.

3. **Decoding process** The actual Viterbi decoding is done at ld_process_raw.

12.3.1 Programs

[Editor Notes: Fix me, the indentation doesn’t work very well.]

```c
if(ld_init(&decoder)) {
    printf("Initialization failed.");
    return -1;
}
if (ld_begin_utt(&decoder, 0)) {
    printf("Cannot start decoding");
    return -1;
}
in_ad = ad_open_sps(samprate);
```
ad_start_rec(in_ad);
while (1) {
    num_samples = ad_read(in_ad, samples, BUFSIZE);
    if (num_samples > 0) {
        /** dump the recorded audio to disk */
        if (ld_process_raw(&decoder, samples, num_samples) < 0) {
            printf("Data processing error.");
            return -1;
        }
    }
}
ad_stop_rec(in_ad);
ad_close(in_ad);
if (ld_end_utt(&decoder)) {
    printf("Cannot end decoding.");
    return -1;
}
ld_finish(&decoder);

Explanation

The above program listing shows a very simple program that could perform the live-mode decoding. For most of the time, the process involve several standard subroutines.

First of all ld_init and ld_finish are functions you used it to do global initialization and termination for the decoder. ld_init would carry out several jobs. First, it reads in the configuration file, then it will also fill in the settings for the decoder structure. At this moment, global resource such as dictionary, acoustic model will be loaded in. They will then be initialized.

ld_begin_utt and ld_end_utt on the other hands carried out sentence initialization and terminataion. This will initialize the necessary information for decoding one utterance.
For every sentence, recorded audio of the speech (by ad_rec) will be continuously feed into ld_process_raw. It is not necessary to use ad_rec. You could use your own audio recording for the application.

Notice that this is a case where no buffering is done in between recording and recognition. This is fine if you could make sure that the recognition time can be done within the recording time. (say less than 1xRT or even better less than 0.5xRT).

If you have doubt about that, you should use a buffering mechanism. Mainly because frames in audio buffer could lost if new audio samples are recorded in the audio device. You can implement a queue or First-in-first-out(FIFO) list to store the frames. Even time, some samples are shifted from the queue and put into processing. In that way, even there is delay, the samples will first store in the queue without begin covered.
12.4 Common problems in setting up the live-mode speech recognizer

Are you doing the right thing? For example, are you sure you recording routine is operated correctly? This can be checked by dumping the audio data to a wave file.

In our experience in CMU, 95 has a problem, it was caused by model/feature mismatch.

What is model/feature mismatch? As you already know, a frame is represented as a vector of floating point number in a speech recognizer. This representation process is called speech encoding. There are many speech encoding schemes if the world. Sphinx use MFCC and Sphinx allow the users a lot of freedom to tweak this encoding scheme.

Here comes the problem, there are several natural places in the training process, you need to specify the parameters of the encoding scheme. For example, in training, when one tries to convert the wave file to cepstrum, one need to specify all these parameters. Now, in decoding, either live-mode or batch mode, if these parameters were not used. There will be dramatic numerical difference between the parameter used in training the model and the parameter used in decoding. In that case, the pattern recognition algorithm will be totally messed up and will not give you any meaningful results.

For one practical example, if you are using the broadcast news model for regression test for our default testing grammar, if there is model/feature mismatch, you will see that the result will become,

<s> </s>

all the time.

Whereas, the correct result is actually

<s> P I T T S B U R G H </s>

Other symptoms of model/feature mismatch includes repeated short outputs is always recognized. Or there is always no output even other parts of the system is diagnosed to be correct.
12.5 livedecodeAPI.c doc generated by doc++
Contents

1 ld_set_uttid — Utility function declarations ............ 238
2 ld_init — Initializes the live-decoder. ..................... 239
3 ld_init_with_args — Initializes the live-decoder. ......... 240
4 ld_finish — Wraps up the live-decoder. ................. 241
5 ld_begin_utt — Marks the start of the current utterance. ................................................... 242
6 ld_end_utt — Marks the end of the current utterance. 243
7 ld_process_raw — Process raw 16-bit samples for the current utterance decoding. ......................... 244
8 ld_process_ceps — Process a buffer of cepstrum frames for the current utterance. ............................ 245
9 ld_retrieve_hyps — Retrieve partial or final decoding results (hypothesis). .................................. 246
10 ld_abort_utt — Abort the current decoding process immediately. ............................................. 247
int ld_set_uttid (live_decoder_t* decoder, char* uttid, int autogen)

Utility function declarations

Utility function declarations
int **ld_init (live_decoder_t* decoder)

Initializes the live-decoder.

Initializes the live-decoder. Internal modules, including the cepstra-generating front-end, the language model, and the acoustic models are initialized, and live-decoder internal variables are set to a starting state.

This version of the live-decoder assumes the user has externally parsed arguments using `<I>cmd_ln_parse()</I>` or `<I>cmd_ln_parse_file()</I>. The user is responsible for calling `<I>cmd_ln_free()</I>` when he/she is done with the live-decoder.

**Return Value:** 0 for success. -1 for failure.

**Parameters:**
- decoder Pointer to the decoder.

**See Also:** `ld_finish` (→4, page 241)
int \texttt{ld\_init\_with\_args} (live\_decoder\_t* decoder, int argc, char** argv)

Initializes the live-decoder.

Initializes the live-decoder. Internal modules, including the cepstrum-generating front-end, the language model, and the acoustic models are initialized, and live-decoder internal variables are set to a starting state.

This version uses the \texttt{cmd\_ln\_h} interface internally. Arguments are parsed and stored internally, and freed later when \texttt{ld\_finish} is called.

\textbf{Return Value:} \hspace{1cm} 0 for success. -1 for failure.

\textbf{Parameters:} \hspace{1cm} decoder Pointer to the decoder.

\hspace{1cm} argc Argument count.

\hspace{1cm} argv Argument string array.

\textbf{See Also:} \hspace{1cm} ld\_finish (→4, \textit{page 241})
int ld_finish (live_decoder_t* decoder)

Wraps up the live-decoder. All internal modules are closed or unloaded. Internal variables are either freed or set to a finishing state. This function should be called once the user is finished with the live-decoder.

**Return Value:** Always return 0 (for success).

**Parameters:**
- decoder Pointer to the decoder.

**See Also:**
- ld_init
- ld_init_with_args (→3, page 240)
int ld_begin_utt (live_decoder_t* decoder, char* uttid)

Marks the start of the current utterance. An utterance is a session of speech decoding that starts with a call to `<I>ld_begin_utt()</I>` and ends with a call to `</I>`. In the duration of an utterance, speech data is processed with either `<I>→ 7, page 244</I>` or `</I>→ 8, page 245`. Decoding results (hypothesis) can be retrieved any time after the start of an utterance using `<I>→ 9, page 246</I>`. However, all previous results will be clobbered at the start of a new utterance.

At the moment, there is an undocumented time limit to the length of an utterance.

**Return Value:**
0 for success. -1 for failure.

**Parameters:**
decoder  Pointer to the decoder.
uttid  Utterance ID string. If `<I>null</I>`, the utterance ID is ignored.

**See Also:**
ld_end_utt
ld_process_raw
ld_process_ceps
ld_retrieve_hyps (`→ 9, page 246`)
int ld_end_utt (live_decoder_t* decoder)

Marks the end of the current utterance. The Live-Decode API can no longer process speech data until the start of the next utterance. Any hypothesis retrieved prior to the end of the utterance is called a partial hypothesis. Any hypothesis retrieved after the end of the utterance is called the final hypothesis. See <I> (→ 9, page 246)</I> on how to retrieve hypothesis.

Return Value: 0 for success. -1 for failure.

Parameters: decoder Pointer to the decoder

See Also: ld_begin_utt
ld_process_raw
ld_process_ceps
ld_retrieve_hyps (→9, page 246)
int `ld_process_raw` (live_decoder_t* decoder, int16* samples, int32 num_samples)

Process raw 16-bit samples for the current utterance decoding.

Process raw 16-bit samples for the current utterance decoding. This function has to be called in the duration of an utterance. That is, in between calls to `<l>` (→ 5, page 242)`<l>` and `<l>` (→ 6, page 243)`<l>`.

**Return Value:** 0 for success. -1 for failure.

**Parameters:**
- `decoder` Pointer to the decoder.
- `samples` Buffer of int16 audio samples.
- `num_samples` Number of samples in the buffer.

**See Also:**
- `ld_begin_utt`
- `ld_end_utt`
- `ld_process_ceps` (→8, page 245)
int **ld_process_cepse (live_decoder_t* decoder, float32** frames, int32 num_frames)

Process a buffer of cepstrum frames for the current utterance.

Process a buffer of cepstrum frames for the current utterance. To use this function, make sure that the parameters to the cepstra-generating front-end that matches the parameters to the decoder’s acoustic model. This function has to be called in the duration of an utterance. That is, in between calls to <I> (→ 5, page 242)</I> and <I> (→ 6, page 243)</I>.

Return Value: 0 for success. -1 for failure.

Parameters:
- decoder Pointer to the decoder.
- frames Buffer of audio feature frames.
- num_frames Number of frames in the buffer.

See Also:
- ld_begin_utt
- ld_end_utt
- ld_process_cepse (→8, page 245)
int ld_retrieve_hyps (live_decoder_t* decoder, char** hyp_str, hyp_t*** hyp_segs)

Retrieve partial or final decoding results (hypothesis).

Retrieve partial or final decoding results (hypothesis). Any hypothesis retrieved prior to the end of the utterance is called a partial hypothesis. Any hypothesis retrieved after the end of the utterance is called the final hypothesis. The hypothesis can be returned in a plain READ-ONLY string and/or an array of READ-ONLY word segments. In the plain string result, all filler and end words are filtered out as well as the pronunciation information. What is left is a very readable string representation of the decoding result. There is no filtering in the word segment result.

Here is an example on how to use the result returned by ld_retrieve_hyps:

<pre>
live_decoder_t d; char *str; hyp_t **segs;
...
ld_retrieve_hyps(&d, &str, &segs); for (; *segs; segs++)
</pre>

Return Value: 0 for success. -1 for failure.

Parameters:

- decoder Pointer to the decoder.
- hyp_str Returned pointer to a READ-ONLY string. If null, the string is not returned.
- hyp_segs Returned pointer to a null-terminated array of word segments. If null, the array is not returned.
int ld_abort_utt (live_decoder_t* decoder)

Abort the current decoding process immediately.

Abort the current decoding process immediately. As opposed to `<I> (→ 6, page 243)</I>`. Retrieving the hypothesis after an abort is not guaranteed.

<EM>!!! NOT IMPLEMENTED YET !!!</EM>

Return Value: 0 for success. -1 for failure.

Parameters: decoder Pointer to the decoder.

See Also: ld_end_utt (→6, page 243)
Chapter 13

Developing Sphinx

Author: Arthur Chan, Editor: Arthur Chan

[Editor Notes: This is more my personal notes on what we should do on development of Sphinx and speech recognition in general.]

This chapter details the reason why sphinx3 and SphinxTrain developed, its underlying philosophy. We will give a very brief introduction of what’s going on in Sphinx3 and SphinxTrain. We will also talk about the future challenges and opportunity of the development.

13.1 Reasons of Developing an Open Source Speech Recognizer

There are two motivations for us to develop a speech recognizer. One is ethical and one is intellectual. They are inter-related.

Let me talk about the intellectual side of our motivation. Speech recognition is very challenging of the problem. There are a lot of things we don’t understand. From human hearing system to how to make computer speech recognition fast enough to work in an embedded device. Being a developer of speech recognition requires vast amount of knowledge in phonetics, linguistics, statistic, pattern recognition and artificial intelligence. Understanding of this knowledge gives us amazing feeling.

Another side of the intellectual problem is how to apply speech recognition. How speech recognizer should interact with human being is still a big problem. For example, why speech-enabled computer games does really
make player having more fun? This is still something many researcher could not explain.

To summarize the above paragraphs in one sentence, it is fun to work on a speech recognizer.

On the ethical side, that might worths more words. At the time of writing, one could only find successful speech recognition application are commercial applications such as dictation, telephone directory system and medical transcription. Many people points out that the success of commercial speech recognition system usually means that users have a poorer service. Many people also rightly pointed out that the benefit of advance of speech recognition could only be enjoyed by a small portion of people in the world.

In user’s forum of Sphinx, we could see people who couldn’t afford the above the technology and would truly benefited by speech recognition. There are people who have been paralyzed because of accident and need speech recognition as the way of input to a computer. There are fathers who want to communicate with his daughter by speech recognition. Not to say, there are a lot of people who could not the tag price of commercial speech recognition. When we look at the current technology, we haven’t done too much for them. We believe that making speech recognition system for these users will truly make a revolution in speech recognition.

We could already see open source software flourish in the world at 2005. Everyone could enjoy the use of GNU/Linux operating system and enjoy high quality and highly stable of system. Emacs and vi are so powerful that expert user could enjoy speed 5 times to 10 times of users of Word or other editors. Apache takes 60-70 web server. As a researcher and developer in speech recognition, developer of Sphinx believe in making speech recognition to be as available as the above open source softwares. We believe in 10-20 years, speech recognition will be truly ubiquitous and high quality speech recognition will mean a true change of human living.

However, one thing that the author feel worry about, most of the state of art research algorithms and code of the recognizer are not open to the general public. Or the users need to pay very high cost to attain license of the recognizer. This may make the users to have less freedom to talk to a computer. And more subtly, they will have less freedom to choose on how they can talk to a computer because usually the grammar and the mode of recognition is predefined.

This is perhaps the reason why we should think our job is an important one. If you share this feeling, you could read on.
13.2 Internal of Sphinx3

13.3 Internal of SphinxTrain

13.4 Challenges of development of Sphinx and open source speech recognition in future

13.4.1 The issue of Model Training

For any one who truly understand the process of speech recognition, one would understand that what hackers are truly lacking right now is a strong databases for open source acoustic models.

Acoustic model and language model, as you have already learned after reading through this manual, is actually the most important factor to determine the performance of a system. The strongest model that the open source community so far is the open source hub-4 model. For one example, it is actually not that good for doing dictation at this point.

To understand the limitation of the hub-4 model, one probably need to understand why it were there in the first place and its development later. It is actually an interesting technology story.

From 1985 to now (2005), the Defense Advanced Research Projects Agency (DARPA) starts to have a series of activities featuring an Olympic-like event for speech recognition. That is called evaluation by many speech people. Evaluation is usually participated by multiple sites and there is a prize which is winner taked all.

The terms you could heard in these days like Resource Management (RM), Wall Street Journal (WSJ) and Air Travel Information System (ATIS), Broadcast News (BN) are actually tasks for the DARPA.

Why DAPRA would like to do it? Major reason is that this research could allow government intelligence agencies to gather information from speech automatically. For this technology will be used for, let me leave it to the reader’s imagination and their leisure reading in Tom Clancy and Dan Brown.

Anyway, CMU joined this process, it participates the RM, the WSJ, the ATIS and at last the BN task. The broad-cast news model is the one that
trained with most amount of training data (140 hours). However, this is only trained for a specific task (only transcribing broadcast news, for a specific metric (word error rate) and for a specific sets of speakers (the one that one could find in the training sets). That explains why it is not the most ideal model for all situations.

For the follow hackers of Sphinx and open source speech recognition, if we are serious in improving this situation, the first thing we should do is to figure out a model that allows us to collaboratively develop stronger acoustic and language models for different situations. The following are the challenges we need to face

- **How do we collect large amount of speech data freely?**
  One good example of doing this is the OpenMind project, could we create some kind of mechanism that collect more or more data?

- **How do we transcribe large amount of speech data freely?**
  It is actually impossible to transcribe a large amount of acoustic data, one possibility is to ask volunteer to rotatively look at different subset of data.

- **How do we efficiently train an acoustic model with large amount of training data?**
  Training with 10 hours of speech would already take up computation of 1 machine for 1 day. If one want to train a serious model, it is necessary to train a model with more than 100 hours of speech. And in future, what we are seeking for is to train model with more than 10000 hours of speech. How could we do this? This future could only be fullfilled by using distributed computing. Could we extend to do something like this?

- **How do we build speaker-specific model?**
  This will allow us to do speaker-specific training or minimally speaker adaptation. Could we build something similar to what ViaVoice has done?\(^1\) I.e. Could we build a user registration system that allow user of Sphinx to register and possibly one could build user specific models.

\(^1\)This idea is suggested by email exchange with me and someone. Please tell me who you are.
13.4.2 The issue of Code duplication

13.4.3 The separation of Sphinx 3 and SphinxTrain

13.4.4 The issue of user interface complexity

13.4.5 The issue of lack of researchers/developers’ Education

13.4.6 Is English the only language?

13.4.7 Is PC the only platform?

13.4.8 Is Sphinx the only recognizer

13.5 How to contribute?

In last section, we have talked about many issues that is

13.5.1 A patch

13.5.2 A customize version

13.5.3 A merge

13.6 For future maintainer
Appendix A

Command Line Information

A.1 Sphinx III Decoders

A.1.1 decode

Tool Description

Example
Command-line Argument Description

Usage: [options]

- **-agc** (Default Value: max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-beam** (Default Value: 1.0e-55)
  Description: beam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value: 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value: Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value: Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value: 1e-80)
  Description: CI phone beam for CI-based GMM Selection. [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value: current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value: 0)
  Description: itional Down-sampling, override normal down sampling.

- **-ctl** (Default Value: Control)
  Description: ile listing utterances to be processed
- **ctlcount** (Default Value: 1000000)
  Description: No. of utterances to be processed (after skipping -ctloffset entries)

- **ctloffset** (Default Value: 0)
  Description: of utterances at the beginning of -ctl file to be skipped

- **ctl_lm** (Default Value: Contr)
  Description: ol file that list the corresponding LM for an utterance

- **ctl_mllr** (Default Value: Cont)
  Description: rol file that list the corresponding MLLR matrix for an utterance

- **dict** (Default Value: Pronunci)
  Description: ation dictionary input file

- **ds** (Default Value: 1)
  Description: Down-sampling the frame computation.

- **epl** (Default Value: 3)
  Description: Per Lextree; #successive entries into one lextree before lextree-entries shifted to the next

- **fdict** (Default Value: Filler)
  Description: word pronunciation dictionary input file

- **feat** (Default Value: 1s_c_d_)
  Description: dd Feature type: Must be s3.1x39 / 1s_c_d_dd/ s2.4x .

- **fillpen** (Default Value: Filler)
  Description: word probabilities input file

- **fillprob** (Default Value: 0.1)
  Description: fault non-silence filler word probability
-gs (Default Value: Gaussian)
  Description: election Mapping.

-gs4gs (Default Value: 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

-hmmdump (Default Value: 0)
  Description: er to dump active HMM details to stderr (for debugging)

-hmmhistbinsize (Default Value: 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

-hyp (Default Value: Recogniti)
  Description: on result file, with only words

-hypseg (Default Value: Recogn)
  Description: ition result file, with word segmentations and scores

-latext (Default Value: lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

-lextreedump (Default Value: 0)
  Description: hether to dump the lextree structure to stderr (for debugging)

-lm (Default Value: Word)
  Description: am language model input file

-lmctlfn (Default Value: Contro)
  Description: l file for language model

(Default Value: NO DEFAULT VALUE)
  Description:
• **-lmdumpdir** (Default Value: The)  
  Description: directory for dumping the DMP file.

• **-lminmemory** (Default Value: 0)  
  Description: ad language model into memory (default: use disk cache for lm)

• **-log3table** (Default Value: 1)  
  Description: ermines whether to use the log3 table or to compute the values at run time.

• **-logbase** (Default Value: 1.0003)  
  Description: Base in which all log-likelihoods calculated

• **-lw** (Default Value: 8.5)  
  Description: e weight

• **-maxhistpf** (Default Value: 100)  
  Description: ax no. of histories to maintain at each frame

• **-maxhmmpf** (Default Value: 20000)  
  Description: Max no. of active HMMs to maintain at each frame; approx.

• **-maxwpf** (Default Value: 20)  
  Description: no. of distinct word exits to maintain at each frame

• **-mdef** (Default Value: Model)  
  Description: finition input file

• **-mean** (Default Value: Mixture)  
  Description: gaussian means input file

• **-mixw** (Default Value: Senone)  
  Description: ixture weights input file
• **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

• **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time

• **-outlatdir** (Default Value : Dire)
  Description: ctory in which to dump word lattices

• **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

• **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

• **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

• **-pl beam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [1 (narrowest)..10000000(very wide)]

• **-pl window** (Default Value : 1)
  Description: ndow size (actually window size-1) of phoneme look-ahead.

• **-ptranskip** (Default Value : 0)
  Description: wbeam for phone transitions every so many frames (if >= 1)

• **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)
- **-silprob** (Default Value : 0.1)
  Description: ault silence word probability

- **-subvq** (Default Value : Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value : 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrowest)]

- **-svq4svq** (Default Value : 0)
  Description: that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value : HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value : 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value : 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-utt** (Default Value : Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value : 0.7)
  Description: weight

- **-var** (Default Value : Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value : 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)
• **-varnorm** (Default Value: no)
  Description: Ance normalize each utterance (yes/no; only applicable if CMN is also performed)

• **-vqeval** (Default Value: 3)
  Description: ue added which used only part of the cepstral vector to do the estimation

• **-wbeam** (Default Value: 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest)..1(narrowest)]

• **-wend** (Default Value: 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

• **-wip** (Default Value: 0.7)
  Description: nsertion penalty

• (Default Value: NO DEFAULT VALUE)
  Description:
A.1.2  livedecode

Tool Description

Example
Command-line Argument Description

Usage: [options]

- **-agc** (Default Value: max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-alpha** (Default Value: 0.97)
  Description: ha for pre-emphasis window

- **-beam** (Default Value: 1.0e-55)
  Description: eam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value: 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value: Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value: Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value: 1e-80)
  Description: CI phone beam for CI-based GMM Selection. Good number should be [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value: current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value: 0)
  Description: itional Down-sampling, override normal down sampling.
- **-ctl** (Default Value: Control)
  Description: file listing utterances to be processed

- **-ctlcoun** (Default Value: 1000000)
  Description: No. of utterances to be processed (after skipping
  -ctlffset entries)

- **-ctlffset** (Default Value: 0)
  Description: of utterances at the beginning of -ctl file to be skipped

- **-ctl_lm** (Default Value: Contr)
  Description: ol file that list the corresponding LMs

- **-dict** (Default Value: Pronunci)
  Description: ation dictionary input file

- **-doublebw** (Default Value: 0)
  Description: her mel filter triangle will have double the bandwidth, 0
  is false

- **-ds** (Default Value: 1)
  Description: Down-sampling the frame computation.

- **-epl** (Default Value: 3)
  Description: Per Lextree; #successive entries into one lextree before
  lextree-entries shifted to the next

- **-fdict** (Default Value: Filler)
  Description: word pronunciation dictionary input file

- **-feat** (Default Value: 1s_c_d)
  Description: dd Feature type: Must be 1s_c_d_dd / s3_1x39 / s2_4x /
  cep_dcep[.

- **-fillpen** (Default Value: Filler)
  Description: word probabilities input file
• **-fillprob** (Default Value: 0.1)
  Description: fault non-silence filler word probability

• **-frate** (Default Value: 100)
  Description: e rate

• **-gs** (Default Value: Gaussian)
  Description: election Mapping.

• **-gs4gs** (Default Value: 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

• **-hmmdump** (Default Value: 0)
  Description: er to dump active HMM details to stderr (for debugging)

• **-hmmhistbinsize** (Default Value: 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

• **-hyp** (Default Value: Recogniti)
  Description: on result file, with only words

• **-hypseg** (Default Value: Recogn)
  Description: ition result file, with word segmentations and scores

• **-input_endian** (Default Value: 0)
  Description: the input data byte order, 0 is little, 1 is big endian

• **-latex** (Default Value: lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

• **-lextreedump** (Default Value: 0)
  Description: hether to dump the lextree structure to stderr (for debugging)
- **-lm** (Default Value: Word)
  Description: AM language model input file

- **-lmctlfn** (Default Value: Control)
  Description: File for language model

- (Default Value: NO DEFAULT VALUE)
  Description:

- **-lmdumpdir** (Default Value: The)
  Description: Directory for dumping the DMP file.

- **-lminmemory** (Default Value: 0)
  Description: Load language model into memory (default: use disk cache for lm)

- **-log3table** (Default Value: 1)
  Description: Determines whether to use the log3 table or to compute the values at run time.

- **-logbase** (Default Value: 1.0003)
  Description: Base in which all log-likelihoods calculated

- **-lowerf** (Default Value: 200)
  Description: Lower edge of filters

- **-lw** (Default Value: 8.5)
  Description: E weight

- **-machine_endian** (Default Value: NO DEFAULT VALUE)
  Description: 0 the machine's endian, 0 is little, 1 is big endian

- **-maxcepvects** (Default Value: 256)
  Description: Maximum number of cepstral vectors that can be obtained from a single sample buffer
• **-maxhistpf** (Default Value : 100)
  Description: ax no. of histories to maintain at each frame

• **-maxhmmpf** (Default Value : 20000)
  Description: Max no. of active HMMs to maintain at each frame; approx.

• **-maxhyplen** (Default Value : 1000)
  Description: Maximum number of words in a partial hypothesis (for block decoding)

• **-maxwpf** (Default Value : 20)
  Description: no. of distinct word exits to maintain at each frame

• **-mdef** (Default Value : Model)
  Description: finition input file

• **-mean** (Default Value : Mixture)
  Description: gaussian means input file

• **-mixw** (Default Value : Senone)
  Description: ixture weights input file

• **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

• **-nfft** (Default Value : 256)
  Description: ts for FFT

• **-nfilt** (Default Value : 31)
  Description: r of mel filters

• **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time
- **-outlatdir** (Default Value : Dire)
  Description: Directory in which to dump word lattices

- **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

- **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

- **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

- **-pl_beam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [0(widest) .. 1(narrowest)]

- **-pl_window** (Default Value : 1)
  Description: Window size (actually window size-1) of phoneme look-ahead.

- **-ptranskip** (Default Value : 0)
  Description: Beam for phone transitions every so many frames (if >= 1)

- **-samprate** (Default Value : 8000)
  Description: Sampling rate (only 8K and 16K currently supported)

- **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)

- **-silprob** (Default Value : 0.1)
  Description: Default silence word probability
- **-subvq** (Default Value: Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value: 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrowest)]

- **-svq4svq** (Default Value: 0)
  Description: g that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value: HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value: 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value: 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-upperf** (Default Value: 3500)
  Description: per edge of filters

- **-utt** (Default Value: Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value: 0.7)
  Description: weight

- **-var** (Default Value: Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value: 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)
• **-varnorm** (Default Value: no)
  Description: normalize each utterance (yes/no; only applicable if CMN is also performed)

• **-vqeval** (Default Value: 3)
  Description: any vectors should be analyzed by VQ when building the shortlist. It speeds up the decoder, but at a cost.

• **-wbeam** (Default Value: 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest)..1(narrowest)]

• **-wend beam** (Default Value: 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

• **-wip** (Default Value: 0.7)
  Description: insertion penalty

• **-wlen** (Default Value: 0.0256)
  Description: window length
A.1.3 livepretend

Tool Description

Example
Command-line Argument Description

Usage: [options]

- **-agc** (Default Value: max)
  Description: tic gain control for c0 ('max' or 'none'); (max: c0 -= max-over-current-sentence(c0))

- **-alpha** (Default Value: 0.97)
  Description: ha for pre-emphasis window

- **-beam** (Default Value: 1.0e-55)
  Description: eam selecting active HMMs (relative to best) in each frame [0(widest)..1(narrowest)]

- **-bghist** (Default Value: 0)
  Description: m-mode: If TRUE only one BP entry/frame; else one per LM state

- **-bptbldir** (Default Value: Direc)
  Description: tory in which to dump word Viterbi back pointer table (for debugging)

- **-cepdir** (Default Value: Input)
  Description: cepstrum files directory (prefixed to filespecs in control file)

- **-ci_pbeam** (Default Value: 1e-80)
  Description: CI phone beam for CI-based GMM Selection. Good number should be [0(widest) .. 1(narrowest)]

- **-cmn** (Default Value: current)
  Description: pstral mean normalization scheme (default: Cep -= mean-over-current-sentence(Cep))

- **-cond_ds** (Default Value: 0)
  Description: itional Down-sampling, override normal down sampling.
• **-ctl** (Default Value : Control)
  Description: ile listing utterances to be processed

• **-ctlcount** (Default Value : 1000000)
  Description: 000 No. of utterances to be processed (after skipping
  -ctloffset entries)

• **-ctloffset** (Default Value : 0)
  Description: of utterances at the beginning of -ctl file to be skipped

• **-ctl_lm** (Default Value : Contr)
  Description: ol file that list the corresponding LMs

• **-dict** (Default Value : Pronunci)
  Description: ation dictionary input file

• **-doublebw** (Default Value : 0)
  Description: her mel filter triangle will have double the bandwidth, 0
  is false

• **-ds** (Default Value : 1)
  Description: Down-sampling the frame computation.

• **-epl** (Default Value : 3)
  Description: Per Lextree; #successive entries into one lextree before
  lextree-entries shifted to the next

• **-fdict** (Default Value : Filler)
  Description: word pronunciation dictionary input file

• **-feat** (Default Value : 1s.c.d.)
  Description: dd Feature type: Must be 1s.c.d.dd / s3.1x39 / s2.4x /
  cep_dcep[],

• **-fillpen** (Default Value : Filler)
  Description: word probabilities input file
• **-fillprob** (Default Value: 0.1)
  Description: fault non-silence filler word probability

• **-frate** (Default Value: 100)
  Description: e rate

• **-gs** (Default Value: Gaussian)
  Description: election Mapping.

• **-gs4gs** (Default Value: 1)
  Description: that specified whether the input GS map will be used for Gaussian Selection. If it is disabled, the map will only provide information to other modules.

• **-hmmdump** (Default Value: 0)
  Description: er to dump active HMM details to stderr (for debugging)

• **-hmmhistbinsize** (Default Value: 5)
  Description: 000 Performance histogram: #frames vs #HMMs active; #HMMs/bin in this histogram

• **-hyp** (Default Value: Recogniti)
  Description: on result file, with only words

• **-hypseg** (Default Value: Recogn)
  Description: ition result file, with word segmentations and scores

• **-input_endian** (Default Value: 0)
  Description: the input data byte order, 0 is little, 1 is big endian

• **-latex** (Default Value: lat.gz)
  Description: Filename extension for lattice files (gzip compressed, by default)

• **-lextreedump** (Default Value: 0)
  Description: hether to dump the lextree structure to stderr (for debugging)
• **-lm** (Default Value: Word)
  Description: AM language model input file

• **-lmctlfn** (Default Value: Contro)
  Description: file for language model

• (Default Value: NO DEFAULT VALUE)
  Description:

• **-lmdumpdir** (Default Value: The)
  Description: directory for dumping the DMP file.

• **-lminmemory** (Default Value: 0)
  Description: ad language model into memory (default: use disk cache for lm

• **-log3table** (Default Value: 1)
  Description: ermines whether to use the log3 table or to compute the values at run time.

• **-logbase** (Default Value: 1.0003)
  Description: Base in which all log-likelihoods calculated

• **-lowerf** (Default Value: 200)
  Description: er edge of filters

• **-lw** (Default Value: 8.5)
  Description: e weight

• **-machine_endian** (Default Value: NO DEFAULT VALUE)
  Description: 0 the machine’s endian, 0 is little, 1 is big endian

• **-maxcepvecs** (Default Value: 256)
  Description: Maximum number of cepstral vectors that can be obtained from a single sample buffer
• **-maxhistpf** (Default Value : 100)
  Description: ax no. of histories to maintain at each frame

• **-maxhmmpf** (Default Value : 20000)
  Description: Max no. of active HMMs to maintain at each frame; approx.

• **-maxhyplen** (Default Value : 1000)
  Description: Maximum number of words in a partial hypothesis (for block decoding)

• **-maxwpf** (Default Value : 20)
  Description: no. of distinct word exits to maintain at each frame

• **-mdef** (Default Value : Model)
  Description: fnition input file

• **-mean** (Default Value : Mixture)
  Description: gaussian means input file

• **-mixw** (Default Value : Senone)
  Description: xture weights input file

• **-mixwfloor** (Default Value : 0.0000)
  Description: 001 Senone mixture weights floor (applied to data from -mixw file)

• **-nfft** (Default Value : 256)
  Description: ts for FFT

• **-nfft** (Default Value : 31)
  Description: r of mel filters

• **-Nlextree** (Default Value : 3)
  Description: of lextrees to be instantiated; entries into them staggered in time
- **-outlatdir** (Default Value : Dire)
  Description: directory in which to dump word lattices

- **-outlatoldfmt** (Default Value : 1)
  Description: Whether to dump lattices in old format

- **-pbeam** (Default Value : 1.0e-50)
  Description: Beam selecting HMMs transitioning to successors in each frame [0(widest)..1(narrowest)]

- **-pheurtype** (Default Value : 0)
  Description: bypass, 1= sum of max, 2 = sum of avg, 3 = sum of 1st senones only

- **-plbeam** (Default Value : 1.0e-80)
  Description: Beam for phoneme look-ahead. [0(widest) .. 1(narrow- est)]

- **-plwindow** (Default Value : 1)
  Description: Window size (actually window size-1) of phoneme look-ahead.

- **-ptranskip** (Default Value : 0)
  Description: Wbeam for phone transitions every so many frames (if >= 1)

- **-samprate** (Default Value : 8000)
  Description: Sampling rate (only 8K and 16K currently supported)

- **-senmgau** (Default Value : .cont.)
  Description: Senone to mixture-gaussian mapping file (or .semi. or .cont.)

- **-silprob** (Default Value : 0.1)
  Description: ault silence word probability
- **-subvq** (Default Value: Sub-vec)
  Description: tor quantized form of acoustic model

- **-subvqbeam** (Default Value: 3.0e-3)
  Description: Beam selecting best components within each mixture Gaussian [0(widest)..1(narrows)]

- **-svq4svq** (Default Value: 0)
  Description: g that specified whether the input SVQ will be used as approximate scores of the Gaussians

- **-tmat** (Default Value: HMM)
  Description: e transition matrix input file

- **-tmatfloor** (Default Value: 0.0001)
  Description: HMM state transition probability floor (applied to -tmat file)

- **-treeugprob** (Default Value: 1)
  Description: TRUE (non-0), Use unigram probs in lextree

- **-upperf** (Default Value: 3500)
  Description: per edge of filters

- **-utt** (Default Value: Utterance)
  Description: file to be processed (-ctlcount argument times)

- **-uw** (Default Value: 0.7)
  Description: weight

- **-var** (Default Value: Mixture)
  Description: aussian variances input file

- **-varfloor** (Default Value: 0.0001)
  Description: Mixture gaussian variance floor (applied to data from -var file)
• **-varnorm** (Default Value: no)
  Description: Ance normalize each utterance (yes/no; only applicable if CMN is also performed)

• **-vqeval** (Default Value: 3)
  Description: Any vectors should be analyzed by VQ when building the shortlist. It speeds up the decoder, but at a cost.

• **-wbeam** (Default Value: 1.0e-35)
  Description: Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

• **-wend beam** (Default Value: 1.0e-)
  Description: 80 Beam selecting word-final HMMs exiting in each frame [0(widest) .. 1(narrowest)]

• **-wip** (Default Value: 0.7)
  Description: Insertion penalty

• **-wlen** (Default Value: 0.0256)
  Description: Window length

• (Default Value: NO DEFAULT VALUE)
  Description:

• (Default Value: NO DEFAULT VALUE)
  Description:
A.1.4 gausubvq

Tool Description

Example
Command-line Argument Description

Usage: [options]

-eps (Default Value : 0.0001)
  Description: stopping criterion: stop iterations if relative decrease in sq(error) < eps

-iter (Default Value : 100)
  Description: number of k-means iterations for clustering

-log3table (Default Value : 1.0003)
  Description: Determines whether to use the log3 table or to compute the values at run time.

-mean (Default Value : Means)
  Description: le

-mixw (Default Value : Mixture)
  Description: weights file (needed, even though it’s not part of the computation)

-mixwfloor (Default Value : 0.0000)
  Description: 001 Floor for non-zero mixture weight values in input model

-stdev (Default Value : 0)
  Description: d.dev. (rather than var) in computing vector distances during clustering

-subvq (Default Value : Output)
  Description: subvq file (stdout if not specified)

-svqrows (Default Value : 4096)
  Description: . of codewords in output subvector codebooks

-svspec (Default Value : Subvec)
  Description: tors specification (e.g., 24,0-11/25,12-23/26-38 or
• -var (Default Value : Variances)
  Description: file

• -varfloor (Default Value : 0.0001)
  Description: Floor for non-zero variance values in input model
A.1.5 align

A.1.6 allphone

A.1.7 dag

A.1.8 astar

A.1.9 cepview

Tool Description

cepview could view the cepstral coefficient create by wave2feat.

Command-line Argument Description
A.2 SphinxTrain: Executables

A.2.1 agg_seg

Tool Description

Description:

agg_seg sample accumulate feature vectors and used it for quantizing the vector space. This functionality is very useful in S2 training initialization. Not all features vectors are used. They are sampled using option -stride.

There are many other options of this command is currently obsolete. Please type -example yes to get a working argument list.

Example

Example:

agg_seg -segdmpdirs segdmpdir -segdmpfn dumpfile -segtype all -ctlfn ctl -cepdir cepdir -cepext .mfc -ceplen 13 -stride 10
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)
-exemple Shows example of how to use the tool (Default Value : no)
-segdmpdirs Segment dump directories (Default Value : NONE)
-segdmpfn Segment dump file (Default Value : NONE)
-segidxfn Segment index into the dump file. (Default Value : NONE)
-segtype Type of segments to dump. all,st,phn (Default Value : st)
-cntfn Per id count file (Default Value : NONE)
-ddcodeext Extension of the VQ 2nd difference cepstrum files (Default Value : xcode)
-lsnfn Lexical transcript file (contains all utts in ctl file) (Default Value : NONE)
-sentdir Root directory of sent (lexical transcript) files (Default Value : NONE)
-sentext Extension of sent (lexical transcript) files (Default Value : NONE)
-ctlfn The control file name (enumerates utts in corpus) (Default Value : NONE)
-mllrctlfn Lists the MLLR transforms for each utterance (Default Value : NONE)
-mllrdir Directory for MLLR matrices (Default Value : NONE)
-nskip The number of utterances to skip in the control file (Default Value : 0)
-runlen The number of utterances to process after skipping (Default Value : -1)
-moddeffn Model definition file containing all the triphones in the corpus. State/transition matrix definitions are ignored. (Default Value : NONE)
-ts2cbfn Tied state to codebook mapping file (may be `.semi.' or `.cont.') (Default Value : NONE)
-cb2mllr fn codebook to MLLR class mapping file (may be ‘.1cls.’) (Default Value : NONE)

-dictfn Lexicon containing all the words in the lexical transcripts. (Default Value : NONE)

-fdictfn Lexicon containing all the filler words in the lexical transcripts. (Default Value : NONE)

-segdir Root directory of the state segmentation files (Default Value : NONE)

-segext Extension of the state segmentation files (Default Value : v8_seg)

-ccodedir Root directory of the VQ cepstrum files (Default Value : NONE)

-ccodeext Extension of the VQ cepstrum files (Default Value : ccode)

-dcodedir Root directory of the VQ difference cepstrum files (Default Value : NONE)

-dcodeext Extension of the VQ cepstrum files (Default Value : d2code)

-pcodedir Root directory of the VQ power files (Default Value : NONE)

-pcodeext Extension of the VQ power files (Default Value : p3code)

-ddcodedir Root directory of the VQ 2nd difference cepstrum files (Default Value : NONE)

-ddcodeext Extension of the VQ 2nd difference cepstrum files (Default Value : xcode)

-cepdir Root directory of the cepstrum files (Default Value : NONE)

-cepext Extension of the cepstrum files (Default Value : mfc)

-ceplen # of coefficients per cepstrum frame (Default Value : 13)

-agc The type of automatic gain control to do max—emax (Default Value : max)

-cmn The do cepstral mean normalization based on current—prior utterance(s) (Default Value : current)

-varnorm Normalize utterance by its variance (Default Value : no)

-silcomp Do silence compression based on current—prior utterance (Default Value : none)

-feat Feature set to compute (Default Value : NONE)
d (Default Value : 4s.12c.24d.3p.12d)

d (Default Value : 1s.12c.12d.3p.12d)

-cachesz  Feature cache size in Mb (Default Value : 200)

-stride  Take every -stride’th frame when producing dmp (Default Value : 1)
A.2.2 bldtree

Tool Description

Description:

Given a set of questions. Build decision tree for a set of feature of a particular phone. By default, decision tree are not built for filler phones and the phone tagged with SIL. One very confusing parameters of this tool is -stwt, if you are training a n-state HMM, you need to specify n values after this flag.

Example

bld_tree -treenf tree -moddefn mdef -mixwfn mixw -meanfn mean -varfn var -psetfn questions -stwt 1.0 0.05 0.01 -state 0 -ssplitmin 1 -ssplitmax 7 -ssplitthr 1e-10 -csplitmin 1 -csplitmax 2000 -csplitthr 1e-10 -cntthresh 10
**Command-line Argument Description**

*Usage: [options]*

- **help**  Shows the usage of the tool (Default Value : no)
- **example**  Shows example of how to use the tool (Default Value : no)
- **treefn**  Name of output tree file to produce (Default Value : NONE)
- **moddeffn**  Model definition file of the discrete models (Default Value : NONE)
- **ts2cbfn**  The type of models to build trees on (Default Value : .semi.)
- **meanfn**  means file for tree building using continuous HMMs (Default Value : NONE)
- **varfn**  variances file for tree building using continuous HMMs (Default Value : NONE)
- **varfloor**  The minimum variance (Default Value : 0.00001)
- **cntthresh**  Ignore all states with counts less than this (Default Value : 0.00001)
- **mixwfn**  PDF’s for tree building using semicontinuous HMMs (Default Value : NONE)
- **psetfn**  phone set definitions for phonetic questions (Default Value : NONE)
- **phone**  Build trees over n-phones having this base phone (Default Value : NONE)
- **state**  Build tree for this state position. E.g. For a three state HMM, this value can be 0,1 or 2. For a 5 state HMM, this value can be 0,1,2,3 or 4, and so on (Default Value : NONE)
- **mwfloor**  Mixing weight floor for tree building (Default Value : 1e-4)
- **stwt**  Weights on neighboring states, This flag needs a string of numbers equal to the number of HMM-states (Default Value : NONE)
- **ssplitthr**  Simple node splitting threshold (Default Value : 8e-4)
- **ssplitmin**  Minimum of simple tree splits to do. (Default Value : 1)
- **ssplitmax**  The maximum number of bifurcations in the simple tree before it is used to build complex questions. (Default Value : 5)
-csplitthr Compound node splitting threshold (Default Value : 8e-4)
-csplitmin Minimum # of compound tree splits to do (Default Value : 1)
-csplitmax Minimum # of compound tree splits to do (Default Value : 100)
A.2.3 bw

Tool Description

Description: Strictly speaking, bw only implements the first-part of the Baum-Welch algorithm. That is it go through forward and backward algorithm and collect the necessary statistics for parameter estimation.

The advantage of this architecture is that researcher can easily write programs to do parameter estimation and they have no need to tweak the huge and usually difficult Baum-Welch algorithm.

In terms of functionality, one important thing you need to know is option -part and -npart. They can allow you to split the training into N equal parts Say, if there are M utterances in your control file, then this will enable you to run the training separately on each (M/N)th part. This flag may be set to specify which of these parts you want to currently train on. As an example, if your total number of parts (-npart) is 3, -part can take one of the values 1, 2 or 3.

To control the speed of the training, -abeam (control alpha search) and -bbeam (control beta search) can be used to control the searching time. Notice that if the beams are too small, the path may not reach the end of the search and results in estimation error Too many lost path may also cause training set likelihood not unable to increase.

Several options allow the user to control the behaviour of bw such that silence or pauses can be taken care. For example. One could use -sildelfn to specify periods of time which was assume to be silence. One could also use -sildel and -siltag to specify a silence and allow them to be optionall deleted.

Finally, one can use the viterbi training mode of the code. Notice though, the code is not always tested by CMU’s researcher

I also included the following paragraph from Rita’s web page. I largely adapted from here and I think it is a pretty good wrap-up of the convergence issues "bw is an iterative re-estimation process, so you have to run many passes of the Baum-Welch re-estimation over your training data. Each of these passes, or iterations, results in a slightly better set of models for the CI phones. However, since the objective function maximized in each of these passes is the likelihood, too many iterations would ultimately result in models which fit very closely to the training data. you might not want this to happen for many reasons. Typically, 5-8 iterations of Baum-Welch are sufficient for getting good estimates of the CI models. You can automatically determine the number of iterations that you need by looking at the total likelihood of the training data at the end of the
first iteration and deciding on a "convergence ratio" of likelihoods. This is simply the ratio of the total likelihood in the current iteration to that of the previous iteration. As the models get more and more fitted to the training data in each iteration, the training data likelihoods typically increase monotonically. The convergence ratio is therefore a small positive number. The convergence ratio becomes smaller and smaller as the iterations progress, since each time the current models are a little less different from the previous ones. Convergence ratios are data and task specific, but typical values at which you may stop the Baum-Welch iterations for your CI training may range from 0.1-0.001. When the models are variance-normalized, the convergence ratios are much smaller."

**Example**

Example: Command used to train continuous HMM (Beware, this only illustrates how to use this command, for detail on how to tune it, please consult the manual.) bw -moddefn mdef -ts2cbfn .cont. -mixwfn mixw -tmatfn tmatn -meanfn mean -varfn var -dictfn dict -fdictfn fillerdict -ctlfn control_files -part 1 -npart 1 -cepdir feature_dir -cepext mfc -lsnfn transcription -accumdir accumdir -abeam 1e-200 -bbeam 1e-200 -meanreest yes -varreest yes -tmatreest yes -feat 1s.12c.12d.3p.12dd -ceplen 13

If you want to do parallel training for N machines. Run N trainers with -part 1 -npart N -part 2 -npart N . . -part N -npart N
Command-line Argument Description

Usage: [options]

-help  Shows the usage of the tool (Default Value : no)

-example  Shows example of how to use the tool (Default Value : no)

-moddeffn  The model definition file for the model inventory to train (Default Value : NONE)

-tmatfn  The transition matrix parameter file name (Default Value : NONE)

-mixwfn  The mixture weight parameter file name (Default Value : NONE)

-meanfn  The mean parameter file name (Default Value : NONE)

-varfn  The var parameter file name (Default Value : NONE)

-mwfloor  Mixing weight smoothing floor (Default Value : 0.00001)

-tpfloor  Transition probability smoothing floor (Default Value : 0.0001)

-varfloor  The minimum variance (Default Value : 0.00001)

-topn  Compute output probabilities based this number of top scoring densities. (Default Value : 4)

-dictfn  The content word dictionary (Default Value : NONE)

-fdictfn  The filler word dictionary (e.g. SIL, SILb, ++COUGH++) (Default Value : NONE)

-ctlfn  The training corpus control file (Default Value : NONE)

-nskip  The number of utterances to skip at the beginning of a control file (Default Value : NONE)

-runlen  The number of utterances to process in the (skipped) control file (Default Value : -1)

-part  Identifies the corpus part number (range 1..NPART) (Default Value : NONE)

-npart  Partition the corpus into this many equal sized subsets (Default Value : NONE)

-cepext  The cepstrum file extension (Default Value : mfc)

-cepdir  The cepstrum data root directory (Default Value : NONE)
-segext State segmentation file extension (Default Value: v8.seg)

-segdir State segmentation file root directory (Default Value: NONE)

-sentdir The sentence transcript file directory (Default Value: NONE)

-sentext The sentence transcript file extension (Default Value: sent)

-lsnfn The corpus word transcript file (Default Value: NONE)

-accumdir A path where accumulated counts are to be written. (Default Value: NONE)

-ceplen The length of the input feature (e.g. MFCC) vectors (Default Value: 13)

-agc The type of automatic gain control to do max—emax (Default Value: max)

-cmn The do cepstral mean normalization based on current—prior utter- ance(s) (Default Value: current)

-varnorm Variance Normalize? (Default Value: no)

-silcomp Do silence compression based on current—prior utterance (Default Value: none)

-sildel Allow optional silence deletion in the Baum-Welch algorithm or the Viterbi algorithm. (Default Value: no)

-siltag Specify the tag of silence, by default it is <sil>. (Default Value: SIL)

-abeam Evaluate alpha values subject to this beam (Default Value: 1e-100)

-bbeam Evaluate beta values (update reestimation sums) subject to this beam (Default Value: 1e-100)

-varreest Reestimate variances (Default Value: yes)

-meanreest Reestimate means (Default Value: yes)

-mixwreest Reestimate mixing weights (Default Value: yes)

-tmatreest Reestimate transition probability matrices (Default Value: yes)

-spkrxfrm A speaker transform to use for SAT modelling (Default Value: NONE)
-mllrmult Reestimate multiplicative term of MLLR adaptation of means (Default Value: no)

-mllradd Reestimate shift term of MLLR adaptation of means (Default Value: no)

-ts2cbfn Tied-state-to-codebook mapping file name (Default Value: NONE)

-feat This argument selects the derived feature computation to use. (Default Value: NONE)

-timing Controls whether profiling information is displayed (Default Value: yes)

-viterbi Controls whether Viterbi training is done (Default Value: no)

-2passvar Reestimate variances based on prior means (Default Value: no)

-sildelfn File which specifies frames of background 'silence' to delete (Default Value: NONE)

-cb2mlrfn Codebook-to-MLLR-class mapping file name (Default Value: NONE)

-spthresh State posterior probability floor for reestimation. States below this are not counted (Default Value: 0.0)

-maxuttlen Maximum # of frames for an utt (0 => no fixed limit) (Default Value: 0)

-ckptintv Checkpoint the reestimation sums every -chkptintv utts (Default Value: NONE)
A.2.4  **cp.parm**

**Tool Description**

Description: Copy parameters such as means and variances from one model to another model. You need to specify a "copy operation file" which each operation looks like this: dest_idx1 src_idx1 dest_idx2 src_idx2 dest_idx3 src_idx3 . . . . For example, the first line will instruct cp_param copy src model index 1 to destination model index 1. This tool is still under heavy development at 20040807

**Example**

Example: This example copy mean from a single file from the source file to 5 mean vector in th destination file. First you need to prepare a file like this: 0 0 1 0 2 0 3 0 4 0

Let's call it cp_op cp.parm -cpopsfn cp_op -igmaufn globalmean -ncbout 5 -ogaufn out.means -feat [Your feature type]
Command-line Argument Description

Usage: [options]

-**help**  Shows the usage of the tool (Default Value : no)
-**example**  Shows example of how to use the tool (Default Value : no)
-**cpopsfn**  Copy operation file name (Default Value : NONE)
-**imixwfn**  Input mixing weight file (Default Value : NONE)
-**omixwfn**  Output mixing weight file (Default Value : NONE)
-**nmixwout** # of mixing weight arrays in the output file (Default Value : NONE)
-**imatfn**  Input transition matrix file (Default Value : NONE)
-**otmatfn**  Output transition matrix file (Default Value : NONE)
-**ntmatout** # of transition matrices in the output file (Default Value : NONE)
-**igaufn**  Input Gaussian density parameter file (Default Value : NONE)
-**ogaufn**  Output Gaussian density parameter file (Default Value : NONE)
-**ncbout** # of codebooks in the output file (Default Value : NONE)
-**feat**  Feature set to use (Default Value : c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1])
A.2.5 delint

Tool Description

Description: (copied from Rita’s web page.)

Deleted interpolation is the final step in creating semi-continuous models. The output of deleted interpolation are semi-continuous models in sphinx-3 format. These have to be further converted to sphinx-2 format, if you want to use the SPHINX-II decoder.

Deleted interpolation is an iterative process to interpolate between CD and CI mixture-weights to reduce the effects of overfitting. The data are divided into two sets, and the data from one set are used to estimate the optimal interpolation factor between CI and CD models trained from the other set. Then the two data sets are switched and this procedure is repeated using the last estimated interpolation factor as an initialization for the current step. The switching is continued until the interpolation factor converges.

To do this, we need *two* balanced data sets. Instead of the actual data, however, we use the Baum-Welch buffers, since the related math is convenient. we therefore need an *even* number of buffers that can be grouped into two sets. DI cannot be performed if you train using only one buffer. At least in the final iteration of the training, you must perform the training in (at least) two parts. You could also do this serially as one final iteration of training AFTER BW has converged, on a non-lsf setup.

Note here that the norm executable used at the end of every Baum-Welch iteration also computes models from the buffers, but it does not require an even number of buffers. BW returns numerator terms and denominator terms for the final estimation, and norm performs the actual division. The number of buffers is not important, but you would need to run norm at the end of EVERY iteration of BW, even if you did the training in only one part. When you have multiple parts norm sums up the numerator terms from the various buffers, and the denominator terms, and then does the division.

Example

delint -accumdirs accumdir -moddeffn mdef -mixwfn mixw -cilambda 0.9 -feat c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/ -ceplen 13 -maxiter 4000
Command-line Argument Description

Usage: [options]

-help  Shows the usage of the tool (Default Value : no)
-example Shows example of how to use the tool (Default Value : no)
-moddeffn The model definition file name (Default Value : NONE)
-mixwfn The mixture weight parameter file name (Default Value : NONE)
-accumdirs A path where accumulated counts are to be read. (Default Value : NONE)
-cilambda Weight of CI distributions with respect to uniform distribution (Default Value : 0.9)
-maxiter max # of iterations if no lambda convergence (Default Value : 100)
-feat 2dd feature stream definition (Default Value : 4s_12c_24d_3p_1)
-ceplen Input feature vector length (e.g. MFCC) (Default Value : 13)
A.2.6 dict2tri

Tool Description

Description: find the triphone list from the dictionary

Example

Example: dict2tri -dictfn dict -basephnfn phonelist -btwtri yes
Command-line Argument Description

Usage: [options]

-**help**  Shows the usage of the tool (Default Value : no)
-**example**  Shows example of how to use the tool (Default Value : no)
-**dictfn**  Dictionary file name (Default Value : NONE)
-**basephnfn**  Base phone list file name (Default Value : NONE)
-**btwtri**  Compute between-word triphone set (Default Value : yes)
A.2.7 inc_comp

Tool Description

Description:

Increase the number of mixture of a continuous HMM. Notice that option -ninc actually means the finally number of mixture one wants to obtain. Usually, it is the power of two. You are also recommended to split the number of mixture from \(1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow \text{and so on.}\)

Example

Example:

```
inc_comp -ninc 16 -dcountfn mixture_weights -inmixwfn mixture_weights -outmixwfn out_mixture_weights -inmeanfn means -outmeanfn out_means -invarfn variance -outvarfn out_variance -ceplen 13
```
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value: no)
-examples Shows example of how to use the tool (Default Value: no)
-ninc The # of densities to split (Default Value: 1)
-inmixwfn The weight file for all N den/mix (Default Value: NONE)
-outmixwfn The output mixing weight file name w/ N+NINC density weights/mixture (Default Value: NONE)
-inmeanfn The source mean file w/ N means (Default Value: NONE)
-outmeanfn The new mean file w/ N+NINC means (Default Value: NONE)
-invarfn The source variance file w/ N means (Default Value: NONE)
-outvarfn The new variance file w/ N+NINC means (Default Value: NONE)
-dcountfn The density counts for the N source den/mix (Default Value: NONE)
-ceplen The length of the input feature (e.g. MFCC) vectors (Default Value: 13)
-feat Defines the acoustic feature set. (Default Value: NONE)
A.2.8 init_gau

Tool Description

Description: (Copy from Rita’s web manual) To initialize the means and variances, global values of these parameters are first estimated and then copied into appropriate positions in the parameter files. The global mean is computed using all the vectors you have in your feature files. This is usually a very large number, so the job is divided into many parts. At this stage you tell the Sphinx how many parts you want it to divide this operation into (depending on the computing facilities you have) and the Sphinx “accumulates” or gathers up the vectors for each part separately and writes it into an intermediate buffer on your machine. The executable init_gau is used for this purpose.

Example

Example:

    init_gau -accumdir accumdir -ctlfn controlfn -part 1 -npart 1 -cepdir cepdir -feat 1s_12c_12d_3p_12dd -ceplen 13
Command-line Argument Description

Usage: [options]

- **help** Shows the usage of the tool (Default Value: no)
- **example** Shows example of how to use the tool (Default Value: no)
- **moddeffn** Model definition file for the single density HMM’s to initialize (Default Value: NONE)
- **ts2cbfn** Tied-state-to-codebook mapping file (Default Value: NONE)
- **accumdir** Where to write mean/var counts (Default Value: NONE)
- **meanfn** Mean file for variance initialization (Default Value: NONE)
- **ctlfn** Control file of the training corpus (Default Value: NONE)
- **nskip** # of lines to skip in the control file (Default Value: NONE)
- **runlen** # of lines to process in the control file (after any skip) (Default Value: NONE)
- **part** Identifies the corpus part number (range 1..NPART) (Default Value: NONE)
- **npart** Partition the corpus into this many equal sized subsets (Default Value: NONE)
- **lsnfn** All word transcripts for the training corpus (consistent order w/-ctlfn!) (Default Value: NONE)
- **dictfn** Dictionary for the content words (Default Value: NONE)
- **fdictfn** Dictionary for the filler words (Default Value: NONE)
- **segdir** Root directory of the training corpus state segmentation files. (Default Value: NONE)
- **segext** Extension of the training corpus state segmentation files. (Default Value: v8_seg)
- **scaleseg** Scale existing segmentation to fit new parameter stream length. (Default Value: no)
- **cepdir** Root directory of the training corpus cepstrum files. (Default Value: NONE)
-cepext Extension of the training corpus cepstrum files. (Default Value: mfc)

-silcomp Controls silence compression. (Default Value: none)

-cmn Controls cepstral mean normalization. (Default Value: current)

-varnorm Controls variance normalization. (Default Value: no)

-age Controls automatic gain control. (Default Value: max)

-feat Controls which feature extraction algorithm is used. (Default Value: NONE)

-ceplen # of components in cepstrum vector (Default Value: 13)
A.2.9 init_mixw

Tool Description

Description: Initialization of mixture weight

Example

Example:

init_mixw -src_moddeffn src_moddeffn -src_ts2cbfn .semi. -src_mixwfn src_mixwfn -src_meanfn src_meanfn -src_varfn src_varfn -src_tmatfn src_tmatfn -dest_moddeffn dest_moddeffn -dest_ts2cbfn .semi. -dest_mixwfn dest_mixwfn -dest_meanfn dest_meanfn -dest_varfn dest_varfn -dest_tmatfn dest_tmatfn -feat c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/ -ceplen 13
**Command-line Argument Description**

Usage: [options]

- **-help** Shows the usage of the tool (Default Value: no)

- **-example** Shows example of how to use the tool (Default Value: no)

- **-src_moddeffn** The source model definition file name (Default Value: NONE)

- **-src_ts2cbfn** The source state definition file name (Default Value: NONE)

- **-src_mixwfn** The source mixing weight file name (Default Value: NONE)

- **-src_meanfn** The source mean file name (Default Value: NONE)

- **-src_varfn** The source variance file name (Default Value: NONE)

- **-src_tmatfn** The source transition matrix file name (Default Value: NONE)

- **-dest_moddeffn** The destination model definition file name (Default Value: NONE)

- **-dest_ts2cbfn** The destination state definition file name (Default Value: NONE)

- **-dest_mixwfn** The destination mixing weight file name (Default Value: NONE)

- **-dest_meanfn** The destination mean file name (Default Value: NONE)

- **-dest_varfn** The destination variance file name (Default Value: NONE)

- **-dest_tmatfn** The destination transition matrix file name (Default Value: NONE)

- **-feat** 2dd Derived feature computation to use (Default Value: 1s_12c_12d_3p_1)

- **-ceplen** Size of the input feature vector length (Default Value: 13)
A.2.10  kmeans_init

Tool Description

Description:
Using the segment dump file generated by external software such as agg_seg to initialize the model. It performs k-mean clustering to create the initial means and variances for s2 hmms. This is an important process of initialization of s2 training.

Example

Example:

kmeans_init -gthobj single -stride 1 -ntrial 1 -minratio 0.001 -ndensity 256 -meanfn outhmm/means -varfnouthmm/variances -reest no -segdmpdirs segmpdir -segdmpfindumpfile -ceplen 13
Command-line Argument Description

Usage: [options]

-help  Shows the usage of the tool (Default Value : no)
-exeample  Shows example of how to use the tool (Default Value : no)
-segdir  Directory containing the state segmentations (Default Value : NONE)
-segext  Extention of state segmentation files (Default Value : v8_seg)
-omoddeffn  Model definition of output models (Default Value : NONE)
-dmoddeffn  Model definition used for observation dump (Default Value : NONE)
-ts2cbfn  Tied-state-to-codebook mapping file (Default Value : NONE)
-lsnfn  LSN file name (word transcripts) (Default Value : NONE)
-dictfn  Dictionary file name (Default Value : NONE)
-fdictfn  Filler word dictionary file name (Default Value : NONE)
-cbcntfn  File containing # of times a codebook ID appears in the corpus
           (Default Value : NONE)
-maxcbobs  Cluster at most this many observations per codebook (Default Value : NONE)
-maxtotobs  Cluster at most approximately this many observations over
           all codebooks (Default Value : NONE)
-featsel  The feature stream (0, 1, ...) to select (Default Value : NONE)
-ctlfn  The training corpus control file (Default Value : NONE)
-cepext  The cepstrum file extension (Default Value : mfc)
-cepdir  The cepstrum data root directory (Default Value : NONE)
-cepleen  The length of the input feature (e.g. MFCC) vectors (Default Value : 13)
-agc  The type of automatic gain control to do max—emax (Default Value : max)
-cmn  The do cepstral mean normalization based on current—prior utter-
       ance(s) (Default Value : current)
-**varnorm** Normalize utterance by its variance (Default Value: no)

-**silcomp** Do silence compression based on current—prior utterance (Default Value: none)

-**feat** 2dd This argument selects the derived feature computation to use. (Default Value: 1s_12c_12d_3p_1)

-**segdmpdirs** segment dmp directories (Default Value: NONE)

-**segdmpfn** segment dmp file (Default Value: NONE)

-**segidxfn** segment dmp index file (Default Value: NONE)

-**fpcachesz** # of file descriptors to cache for observation dmp files (Default Value: 3000)

-**obscachesz** # of Mbytes cache to use for observations (Default Value: 92)

-**ndensity** # of densities to initialize per tied state per feature (Default Value: NONE)

-**ntrial** random initialized K-means: # of trials of k-means w/ random initialization from within corpus (Default Value: 5)

-**minratio** K-means: minimum convergence ratio, (p_squerr - squerr) / p_squerr (Default Value: 0.01)

-**maxiter** K-means: maximum # of iterations of updating to apply (Default Value: 100)

-**mixwfn** Output file for mixing weights (Default Value: NONE)

-**meanfn** Output file for means (Default Value: NONE)

-**varfn** Output file for variances (Default Value: NONE)

-**method** Initialization method. Options: rkm — fnkm (Default Value: rkm)

-**reest** Reestimate states according to usual definitions assuming vit seg. (Default Value: yes)

-**niter** # of iterations of reestimation to perform per state (Default Value: 20)

-**gthobj** Gather what kind of obj state, phn, frame (Default Value: state)

-**stride** Gather every -stride’th frame (Default Value: 32)

-**runlen** # of utts to process from ctl file (Default Value: NONE)
-tsoff  Begin tied state reestimation with this tied state (Default Value : 0)

-tscnt  Reestimate this many tied states (if available) (Default Value : NONE)

-tsrngfn  The range of tied states reestimated expressed as offset,count (Default Value : NONE)

-vartiethr  Tie variances if # of observations for state exceed this number (Default Value : 0)
A.2.11 map_adapt

**Tool Description**

**Description:**

Given a speaker-independent (or other baseline) model and a maximum-likelihood estimate of model parameters from adaptation data, map_adapt will update (interpolate) the mean vectors to maximize the a posteriori probability of the adaptation data.

The ML estimate is generated by running a single iteration of the forward-backward algorithm on the adaptation data, using the baseline models as the initial estimate. This can be accomplished using the programs 'bw' and 'norm'. The -mlmeanfn and -mlvarfn arguments are the output parameter files created by 'norm'. The -mlcountfn argument is the file containing observation counts, which is generated with the -dcountfn argument to 'norm'.

**Example**

Example: map_adapt -mapmeanfn map_model/means -mlmeanfn ml_model/means -mlvarfn ml_model/variances -simeanfn baseline/means -sivarfn baseline/variances -mlcntfn bwac-cumdir/gauden_counts
**Command-line Argument Description**

*Usage: [options]*

- **-help** Shows the usage of the tool (Default Value: no)
- **-example** Shows example of how to use the tool (Default Value: no)
- **-mapmeanfn** The output MAP mean file (Default Value: NONE)
- **-simeanfn** The input speaker-independent (baseline) mean file (Default Value: NONE)
- **-sivarfn** The input speaker-independent (baseline) var file (Default Value: NONE)
- **-mlmeanfn** The input ML mean file (output of ‘norm’ on adaptation data) (Default Value: NONE)
- **-mlvarfn** The input ML var file (output of ‘norm’ on adaptation data) (Default Value: NONE)
- **-mlcntfn** The ML total observation count (output of ‘norm -dcountfn’) file (Default Value: NONE)
- **-dnom.weight** The prior adaptation weight (Default Value: 1.0)
A.2.12 make_quests

Tool Description

Description:

This is an implementation of Dr. Rita Singh’s automatic question generation. (Copied from Rita’s comment) The current algorithm clusters CI distributions using a hybrid bottom-up top-down clustering algorithm to build linguistic questions for decision trees. (From Arthur: I need to do some tracing before understand it what’s the internal of the code)

Example

Example: make_quest -moddeffn mdef -meanfn mean -varfn var -mixwfn mixwfn -npermute 8 -niter 1 -qstperstt 20 -tempfn temp -questfn questions
Command-line Argument Description

Usage: [options]

-help  Shows the usage of the tool (Default Value : no)

-example  Shows example of how to use the tool (Default Value : no)

-moddeffn  Model definition file of the ci models (Default Value : NONE)

-meanfn  means file for tree building using continuous HMMs (Default Value : NONE)

-varfn  variances file for tree building using continuous HMMs (Default Value : NONE)

-varfloor  The minimum variance (Default Value : 1.0e-08)

-mixwfn  PDF’s for tree building using semicontinuous HMMs (Default Value : NONE)

-npermute  The minimum variance (Default Value : 6)

-niter  Number of iterations (Default Value : 0)

-qstperstt  something per state (Default Value : 8)

-tempfn  File to write temporary results to (important) (Default Value : /tmp/TEMP.QUESTS)

-questfn  File to write questions to (Default Value : NONE)

-type  HMM type (Default Value : NONE)
A.2.13 mixw_interp

**Tool Description**

Description:

A routine that provides an ad-hoc way of speaker adaptation by mixture weight interpolation. SD and SI model's mixture weight are first determined and they act as inputs of this program. The output is the interpolated mixture weight.

The interpolation is controlled by the value lambda (-sillambda)

**Example**

Example:

```
mixw_interp -SImixwfn si_mixw -SDmixwfn sd_mixw -outmixwfn final_mixw -Sillambda 0.7
```
**Command-line Argument Description**

*Usage: [options]*

- **help**  Shows the usage of the tool (Default Value : no)
- **example**  Shows example of how to use the tool (Default Value : no)
- **SI\text{mixwfn}**  The SI mixture weight parameter file name (Default Value : NONE)
- **SD\text{mixwfn}**  The SD mixture weight parameter file name (Default Value : NONE)
- **out\text{mixwfn}**  The output interpolated mixture weight parameter file name (Default Value : NONE)
- **SI\text{lambda}**  Weight given to SI mixing weights (Default Value : 0.5)
A.2.14  mk\_flat

**Tool Description**

Description: (Copied from Rita's web page) In flat-initialization, all mixture weights are set to be equal for all states, and all state transition probabilities are set to be equal. Unlike in continuous models, the means and variances of the codebook Gaussians are not given global values, since they are already estimated from the data in the vector quantization step. To flat-initialize the mixture weights, each component of each mixture-weight distribution of each feature stream is set to be a number equal to 1/N, where N is the codebook size. The mixture_weights and transition_matrices are initialized using the executable mk\_flat.

**Example**

Example:

```
  mk\_flat -moddeffn CFS3.ci.mdef -topo CFS3.topology -mixwfn mixture_weights -tmatfn transition_matrices -nstream 1 -ndensity 1
```
**Command-line Argument Description**

*Usage: [options]*

- **-help** Shows the usage of the tool (Default Value: no)
- **-example** Shows example of how to use the tool (Default Value: no)
- **-moddeffn** A SPHINX-III model definition file name (Default Value: NONE)
- **-mixwfn** An output SPHINX-III mixing weight file name (Default Value: NONE)
- **-topo** A template model topology matrix file (Default Value: NONE)
- **-tmatfn** An output SPHINX-III transition matrix file name (Default Value: NONE)
- **-nstream** Number of independent observation streams (Default Value: 4)
- **-ndensity** Number of densities per mixture density (Default Value: 256)
A.2.15  mk_mdef_gen

Tool Description

Description:

(Copied from Rita’s comment and I think it is a pretty good description.) Multi-function routine to generate mdef for context-independent training, untied training, and all-triphones mdef for state tying. Flow: if (triphonelist) make CI phone list and CD phone list if alltriphones mdef needed, make mdef if (rawphonelist) Make ci phone list, if cimdef needed, make mdef Generate alltriphones list from dictionary if alltriphones mdef needed, make mdef if neither triphonelist or rawphonelist quit Count triphones and triphone types in transcript Adjust threshold according to min-occ and maxtriphones Prune triphone list Make untied mdef

Example

Example: Create CI model definition file mk_mdef_gen -phnlstfn phonefile -ocimdef ci_mdeffile -n_state_pm 3

Create untied CD model definition file mk_mdef_gen -phnlstfn rawphonefile -dictfn dict -fdictfn filler_dict -lsnfn transcription -ountiedmdef untie_mdef -n_state_pm 3 -maxtriphones 10000 Create tied CD model definition file mk_mdef_gen -phnlstfn rawphone -oalltphnmdef untie_mdef -dictfn dict -fdictfn filler_dict -n_state_pm 3.
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-phnlstfn List of phones (Default Value : NONE)

-inCImdef Input CI model definition file. (Default Value : NONE)

-noed (Default Value : If)

(inCImdef ignored (Default Value : If)

(inCImdef ignored (Default Value : NONE)

-inCDmdef Input CD model definition file. (Default Value : NONE)

-inCDmdef ignored (Default Value : If)

(inCImdef ignored (Default Value : NONE)

-dictfn Dictionary (Default Value : NONE)

-fdictfn Filler dictionary (Default Value : NONE)

-lnfn Transcripts file (Default Value : NONE)

-n_state_pm No. of states per HMM (Default Value : 3)

-ocountfn Output phone and triphone counts file (Default Value : NONE)

-oCImdef Output CI model definition file (Default Value : NONE)

-oalltphnmdfn Output all triphone model definition file (Default Value : NONE)

-ountiedmdef Output untied model definition file (Default Value : NONE)

-minocc Min occurances of a triphone must occur for inclusion in mdef file (Default Value : 1)

-maxtriphones Max. number of triphones desired in mdef file (Default Value : 100000)
A.2.16  mk_mllr_class

**Tool Description**

Description: Create the senone to mllr class mapping.

**Example**

Example: `mk_mllr_class -nmap mapfile -nclass 4 -cb2mllrfn out.cb2mllr`
Command-line Argument Description

Usage: [options]

- **help** Shows the usage of the tool (Default Value: no)
- **example** Shows example of how to use the tool (Default Value: no)
- **nmap** # of codebook -> MLLR class mappings (Default Value: NONE)
- **nclass** # of MLLR classes to map into (Default Value: NONE)
- **cb2mllrfn** Codebook-to-MLLR mapping file to create (Default Value: NONE)
A.2.17  mk_model_def

**Tool Description**

Description:

(Copied from Eric Thayers' comment) Make SPHINX-III model definition files from a variety input sources. One input source is a SPHINX-II senone mapping file and triphone file. Another is a list of phones (in which case the transition matrices are tied within base phone class and the states are untied)

**Example**

Example: [Under construction]
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn A SPHINX-III model definition file name (Default Value : NONE)

-triponefn A SPHINX-II triphone file name (Default Value : NONE)

-phonelstfn List of phones (first 4 columns of phone def'ns) (Default Value : NONE)

-mapfn A SPHINX-II senone mapping file name (Default Value : NONE)

-n_cdstate total # of CD senones/tied_states (even though mapfn may reference fewer) (Default Value : NONE)

-n_cistate # of CI senones/tied_states (Default Value : NONE)

-n_tmat # of tied transition matrices (Default Value : NONE)

-n_state_pm # of states/model (Default Value : 5)
A.2.18  mk_s2cb

Tool Description

Description:
Convert s3 means and variances to s2 codebook format. cepstrum, delta, delta-delta and power cepstrum basename could be changed by users.

Example

Example:

    mk_s2cb -meanfn s3mean -varfn s3var -cbdir s2dir -varfloor 0.00001
Command-line Argument Description

Usage: [options]

- **help** Shows the usage of the tool (Default Value : no)
- **example** Shows example of how to use the tool (Default Value : no)
- **meanfn** A SPHINX-III mean density parameter file name (Default Value : NONE)
- **varfn** A SPHINX-III variance density parameter file name (Default Value : NONE)
- **cbdir** A directory containing SPHINX-II 1PD codebooks (Default Value : NONE)
- **varfloor** Minimum variance value (Default Value : NONE)
- **cepcb** Basename of the cepstrum codebook (Default Value : cep.256)
- **dcepcb** Basename of the difference cepstrum codebook (Default Value : d2cep.256)
- **powcb** Basename of the power codebook (Default Value : p3cep.256)
- **2dcepcb** Basename of the 2nd order difference cepstrum codebook (Default Value : xcep.256)
- **meanext** Mean codebook extension (Default Value : vec)
- **varext** Variance codebook extension (Default Value : var)
A.2.19  mk_s2hmm

Tool Description

Description:

Convert s3 model definition file, mixture weight and transition matrices to s2 hmm format.

Example

Example:

mk_s2hmm -meanfn s3mdef -varfn s3mixw -tmat s3tmat -hmmdir s2dir
Command-line Argument Description

Usage: [options]

**-help** Shows the usage of the tool (Default Value : no)

**-example** Shows example of how to use the tool (Default Value : no)

**-moddeffn** Model definition file for the S3 models (Default Value : NONE)

**-tmatfn** S3 transition matrix parameter file name (Default Value : NONE)

**-tpfloor** Transition probability smoothing floor (Default Value : 0.0001)

**-mixwfn** S3 mixing weight parameter file name (Default Value : NONE)

**-hmmdir** S2 model output directory (Default Value : NONE)

**-hmmext** Extension of a SPHINX-II model file. (Default Value : chmm)

**-cepsenoext** Extension of cepstrum senone weight file (Default Value : ccode)

**-dcepsenoext** Extension of difference cepstrum senone weight file (Default Value : d2code)

**-powsenoext** Extension of power senone weight file (Default Value : p3code)

**-2dcepsenoext** Extension of 2nd order difference cepstrum senone weight file (Default Value : xcode)

**-mtype** Model type sdm,dhmm (Default Value : sdm)
A.2.20 mk_s2phone

Tool Description

Description:
(Copied from Eric Thayer's comment) * Make a SPHINX-II phone file given a list of phones (in SPHINX-III * format). Ok, this sounds like a bizarre fn, but I did need it * once.

Example

Example:

mk_s2phone -s2phonefn s2.phone -phonelstfn s3.phone
Command-line Argument Description

Usage: [options]

-**help**  Shows the usage of the tool (Default Value : no)
-**example**  Shows example of how to use the tool (Default Value : no)
-**s2phonefn**  A SPHINX-II phone file name (Default Value : NONE)
-**phonelstfn**  List of phones (Default Value : NONE)

(Default Value : NONE)
A.2.21  mk_s2phonemap

Tool Description

Description:

Convert s3 model definition file to s2 phone and map files.

Example

Example:

  mk_s2phonemap -moddeffn s3mean -phonefn s2/phone -mapfn s2/map
**Command-line Argument Description**

*Usage: [options]*

- **-help** Shows the usage of the tool (Default Value : no)
- **-example** Shows example of how to use the tool (Default Value : no)
- **-moddeffn** The model definition file for the model to be converted (Default Value : NONE)
- **-phonefn** Output phone file name (Default Value : NONE)
- **-mapfn** Output map file name (Default Value : NONE)
A.2.22 mk_s2sendump

**Tool Description**

Description: Convert s3 model definition file and s3 mixture weight file to a s2 senddump file.

**Example**

Example:

```
   mk_s2sendump -moddeffn s3mdef -mixwfn s3mixw -tpfloor 0.0000001
   -feattype s2_4x -sendumpfn s2dir/sendump
```
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value: no)

-example Shows example of how to use the tool (Default Value: no)

-moddeffn The model definition file for the model inventory to train (Default Value: NONE)

-mixwfn The mixture weight parameter file name (Default Value: NONE)

-sendumpfn Output sendump file name (Default Value: NONE)

-feattype Feature type e.g. s2.4x (Default Value: NONE)

-tpfloor Transition probability smoothing floor (Default Value: 0.0001)
A.2.23  mk_s3gau

**Tool Description**

Description: Conversion from sphinx 2 codebook to sphinx3 means and variances

**Example**

Example: `mk_s3gau -meanfn s3mean -varfn s3var -cbdir s2dir -feat 4s_12c_24d_3p_12dd`
Command-line Argument Description

Usage: [options]

-**help** Shows the usage of the tool (Default Value : no)

-**example** Shows example of how to use the tool (Default Value : no)

-**meanfn** A SPHINX-III mean density parameter file name (Default Value : NONE)

-**varfn** A SPHINX-III variance density parameter file name (Default Value : NONE)

-**cbdir** A directory containing SPHINX-II 1PD codebooks (Default Value : NONE)

-**varfloor** Minimum variance value (Default Value : NONE)

-**cepcb** Basename of the cepstrum codebook (Default Value : cep.256)

-**dcepcb** Basename of the difference cepstrum codebook (Default Value : d2cep.256)

-**powcb** Basename of the power codebook (Default Value : p3cep.256)

-**2dcepcb** Basename of the 2nd order difference cepstrum codebook (Default Value : xcep.256)

-**meanext** Mean codebook extension (Default Value : vec)

-**varext** Variance codebook extension (Default Value : var)

-**fixpowvar** Fix the power variance to the SPHINX-II standards (Default Value : false)

-**feat** 2dd Defines the feature set to use (Default Value : 4s_12c_24d_3p_1)

-**ceplen** Defines the input feature vector (e.g. MFCC) len (Default Value : 13)
A.2.24  **mk_s3mix**

**Tool Description**

Description: Conversion from sphinx 2 hmms to sphinx3 mixture weights

**Example**

Example: (By Arthur: Not sure, may be obsolete) mk_s3mixw -mixwfn s3mixw -moddeffn s3mdef -hmmdir s2hmmmdir
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value: no)

-example Shows example of how to use the tool (Default Value: no)

-mixwfn A SPHINX-III mixture weight parameter file name (Default Value: NONE)

-moddeffn A SPHINX-III model definition file name (Default Value: NONE)

-floor Floor weight value to apply before renormalization (Default Value: NONE)

-ci2cd CD weights initialized to CI weights (-hmmdir is to a set of CI models) (Default Value: false)

-hmmdir A directory containing SPHINX-III models consistent with -moddeffn (Default Value: NONE)

-cepsenoext Extension of cepstrum senone weight file (Default Value: ccode)

-dcepsenoext Extension of difference cepstrum senone weight file (Default Value: d2code)

-powsenoext Extension of power senone weight file (Default Value: p3code)

-2dcepsenoext Extension of 2nd order difference cepstrum senone weight file (Default Value: xcode)
A.2.25  *mk_s3tmat*

**Tool Description**

**Example**
Command line Argument Description
A.2.26 mk_ts2cb

**Tool Description**

Description: (copied from Eric’s comments) * Create a tied-state-to-codebook mapping file for semi-continuous, * phone dependent or fully continuous Gaussian density tying.

**Example**

Example: (By Arthur: Not sure, may be obsolete) mk_ts2cb -moddeffn semi -ts2cbfn ts2cb
Command-line Argument Description

Usage: [options]

-help  Shows the usage of the tool (Default Value : no)
-example  Shows example of how to use the tool (Default Value : no)
-ts2cbfn  A SPHINX-III tied-state-to-cb file name (Default Value : NONE)
-moddeffn  A SPHINX-III model definition file name (Default Value : NONE)
-tyingtype  Output a state parameter def file for fully continuous models
  (Default Value : semi)
-pclassfn  A SPHINX-II phone class file name (Default Value : NONE)
A.2.27  mllr\_solve

**Tool Description**

Description:

Given a set of mean accumulator, mllr\_solve can compute the transform matrix based on the maximum likelihood criteria.

The mean and variance are required to be input in arguments -meanfn and -varfn For linear regression equation y=Ax+b, If you specific only -mllrmult, then only A will be estimated. If you specific only -mllradd, then only b will be estimated.

**Example**

Example: The simplest case: mllr\_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var

Adapt based on only CD-senones mllr\_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var -cdonly yes -moddeffn mdef.

Only adapt on A : mllr\_solve -outmllrfn output.matrix -accumdir accumdir -meanfn mean -varfn var -mllrmult yes -mllradd no

help and example: mllr\_solve -help yes -example yes
Command-line Argument Description

Usage: [options]

-**help** Shows the usage of the tool (Default Value : no)

-**example** Shows example of how to use the tool (Default Value : no)

-**outmllrfn** Output MLLR regression matrices file (Default Value : NONE)

-**accumdir** Paths containing reestimation sums from bw (Default Value : NONE)

-**meanfn** Baseline Gaussian density mean file (Default Value : NONE)

-**varfn** variance (baseline-var, or error-var) file (Default Value : NONE)

-**cb2mllrfn** Codebook to mllr class mapping index file (If it is given, ignore -cdonly) (Default Value : .1cls.)

-**cdonly** Use only CD senones for MLLR (If yes, -moddeffn should be given.) (Default Value : no)

-**moddeffn** Model Definition file (to get CD starting point for MLLR) (Default Value : NONE)

-**mllrmult** Reestimate full multiplicative term of MLLR adatpation of means (yes/no) (Default Value : yes)

-**mllradd** Reestimate shift term of MLLR adaptation of means (yes/no) (Default Value : yes)

-**varfloor** var floor value (Default Value : 1e-3)
A.2.28  mllr_transform

Tool Description

Description:

Given a set of MLLR transform, mllr_transform can transform the mean according to formula \( y = Ax + b \).

The output and input files are specified by -outmeanfn and -inmeanfn respectively. You may also transform the context-dependent model using the option -cdonly. In that case you need to specify a model definition using -moddeffn.

Example

Example: The simplest case: mllr_transform -inmeanfn inMeans -outmeanfn outMeans -mllrmat matrix

    Adapt only on CD phones: mllr_transform -inmeanfn inMeans -outmeanfn outMeans -mllrmat matrix -cdonly yes -moddeffn mdef

    Help and example: nmllr_transform -help yes -example yes
**Command-line Argument Description**

*Usage: [options]*

- **-help** Shows the usage of the tool (Default Value: no)
- **-example** Shows example of how to use the tool (Default Value: no)
- **-inmeanfn** Input Gaussian mean file name (Default Value: NONE)
- **-outmeanfn** Output Gaussian mean file name (Default Value: NONE)
- **-mllrmat** The MLLR matrix file (Default Value: NONE)
- **-cb2mllrfn** The codebook-to-MLLR class file. Override option -cdonly (Default Value: .1cls.)
- **-cdonly** Use CD senones only. -moddeffn must be given. (Default Value: no)
- **-varfloor** Value of the variance floor. Mainly for smoothing the mean. (Default Value: 1e-2)
- **-moddeffn** Model Definition file. (Default Value: NONE)
- **-varfn** Gaussian variance file name. For smoothing. (Default Value: NONE)

(Default Value: NONE)
A.2.29  norm

Tool Description

Description: compute the HMM's parameter generated by bw.

Example

Example: norm -accumdir accumdir -mixwfn mixw -tmatfn tmat -meanfn mean -varfn variances
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-accumdir Paths containing reestimation sums from bw (Default Value : NONE)

-oaccumdir Path to contain the overall reestimation sums (Default Value : NONE)

-tmatfn Transition prob. matrix file to produce (if any) (Default Value : NONE)

-mixwfn Mixing weight file to produce (if any) (Default Value : NONE)

-meanfn Gaussian density mean file to produce (if any) (Default Value : NONE)

-varfn Gaussian density variance file to produce (if any) (Default Value : NONE)

-regmatfn MLLR regression matrices file to produce (if any) (Default Value : NONE)

-dcountfn Gaussian density count file to produce (Default Value : NONE)

-inmixwfn Use mixing weights from this file if never observed (Default Value : NONE)

-inmeanfn Use mean from this file if never observed (Default Value : NONE)

-invarfn Use var from this file if never observed (Default Value : NONE)

-feat The feature set to use. (Default Value : NONE)

-ceplen The vector length of the source features (e.g. MFCC) (Default Value : NONE)
A.2.30  **param_cnt**

**Tool Description**

Description:

Find the number of times each of the triphones listed in a given model definition file (by `-moddeffn`) occurred in a set of transcription, specified by `-lsnfn`.

**Example**

Example: `param_cnt -moddeffn mdef -ts2cbfn .cont. -ctlfn controlfile -lsnfn transcripts -dictfn dict -fdictfn fillerdict -paramtype phone`
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)

-example Shows example of how to use the tool (Default Value : no)

-moddeffn Model definition file for the single density HMM’s to initialize (Default Value : NONE)

-ts2cbfn Tied-state-to-codebook mapping file (Default Value : NONE)

-ctlfn Control file of the training corpus (Default Value : NONE)

-part Identifies the corpus part number (range 1..NPART) (Default Value : NONE)

-npart Partition the corpus into this many equal sized subsets (Default Value : NONE)

-nskip # of lines to skip in the control file (Default Value : NONE)

-runlen # of lines to process in the control file (after any skip) (Default Value : NONE)

-lsnfn All word transcripts for the training corpus (consistent order w/-ctlfn!) (Default Value : NONE)

-dictfn Dictionary for the content words (Default Value : NONE)

-fdictfn Dictionary for the filler words (Default Value : NONE)

-segdir Root directory of the training corpus state segmentation files. (Default Value : NONE)

-segext Extension of the training corpus state segmentation files. (Default Value : v8_seg)

-paramtype Parameter type to count 'state', 'cb', 'phone' (Default Value : state)
A.2.31 printp

**Tool Description**

Description:

Display numerical values of resources generated by Sphinx. Currently we support the following formats:

- `tmatfn`: transition matrix
- `mixwfn`: mixture weight file
- `gaufn`: mean or variance
- `gaucntn`: sufficient statistics for mean and diagonal covariance
- `lambdafn`: interpolation weight

Currently, some parameters can be specified as intervals such as mixture weight.

You can also specify `-sigfig` the number of significant digits by you would like to see.

and normalize the parameters by `-norm`

**Example**

Example:

Print the mean of a Gaussian: `printp -gaufn mean`

Print the variance of a Gaussian: `printp -gaufn var`

Print the sufficient statistic: `printp -gaucntfn gaucnt`

Print the mixture weights: `printp -mixw mixw`

Print the interpolation weight: `printp -lambdafn lambda`
**Command-line Argument Description**

*Usage: [options]*

- **help**  Shows the usage of the tool (Default Value : no)
- **example**  Shows example of how to use the tool (Default Value : no)
- **tmatfn**  The transition matrix parameter file name (Default Value : NONE)
- **mixwfn**  The mixture weight parameter file name (Default Value : NONE)
- **mixws**  Start id of mixing weight subinterval (Default Value : NONE)
- **mixwe**  End id of mixing weight subinterval (Default Value : NONE)
- **gaufn**  A Gaussian parameter file name (either for means or vars) (Default Value : NONE)
- **gaucntfn**  A Gaussian parameter weighted vector file (Default Value : NONE)
- **regmatcntfn**  MLLR regression matrix count file (Default Value : NONE)
- **moddeffn**  The model definition file (Default Value : NONE)
- **lambdafn**  The interpolation weight file (Default Value : NONE)
- **lambdamin**  Print int. wt. >= this (Default Value : 0)
- **lambdamax**  Print int. wt. <= this (Default Value : 1)
- **norm**  Print normalized parameters (Default Value : yes)
- **sigfig**  Number of significant digits in 'e' notation (Default Value : 4)
A.2.32 prunetree

Tool Description

Description: Using prunetree, the bifurcations in the decision trees which resulted in the minimum increase in likelihood are progressively removed and replaced by the parent node. The selection of the branches to be pruned out is done across the entire collection of decision trees globally.

Example

Example:

    prunetree -itreedir input_tree_dir -nseno 5000 -otreedir output_tree_dir -moddefn mdef -psetfn questions -minocc 100
Command-line Argument Description

Usage: [options]

- **help**  Shows the usage of the tool (Default Value : no)
- **example**  Shows example of how to use the tool (Default Value : no)
- **moddeffn**  CI model definition file (Default Value : NONE)
- **psetfn**  Phone set definition file (Default Value : NONE)
- **itreedir**  Input tree directory (Default Value : NONE)
- **otreedir**  Output tree directory (Default Value : NONE)
- **nseono**  # of senones defined by the output trees (Default Value : NONE)
- **minocc**  Prune nodes w/ fewer than this # of observations (Default Value : 0.0)
A.2.33  tiestate

Tool Description

Description : Create a model definition file with tied state from model definition file without tied states.

Example

Example: tiestate -imoddeffn  imdef -omoddeffn  omdef -treedir  trees -psetfn questions

This is an example of the input and output format, I copied from Rita's web page,

This is an hypothetical input of tiestate # triphone: (null) # seno map: (null) # 0.3 5 n_base 34 n_tri 156 n_state_map 117 n_tied_state 15 n_tied_ci_state 5 n_tied_tmat # # Columns definitions #base lft rt p attrib tmat ... state id's ... SIL - - - filler 0 0 1 2 N AE - - - n/a 1 3 4 5 N AX - - - n/a 2 6 7 8 N B - - - n/a 3 9 10 11 N T - - - n/a 4 12 13 14 N AE B T i n/a 1 15 16 17 N AE T B i n/a 1 18 19 20 N AX AX AX s n/a 2 21 22 23 N AX AX B s n/a 2 24 25 26 N AX AX SIL s n/a 2 27 28 29 N AX AX T s n/a 2 30 31 32 N AX B AX s n/a 2 33 34 35 N AX B B s n/a 2 36 37 38 N AX B SIL s n/a 2 39 40 41 N AX B T s n/a 2 42 43 44 N AX SIL AX s n/a 2 45 46 47 N AX SIL B s n/a 2 48 49 50 N AX SIL SIL s n/a 2 51 52 53 N AX SIL T s n/a 2 54 55 56 N AX T AX s n/a 2 57 58 59 N AX T B s n/a 2 60 61 62 N AX T SIL s n/a 2 63 64 65 N AX T T s n/a 2 66 67 68 N B AE AX e n/a 3 69 70 71 N B AE B e n/a 3 72 73 74 N B AE SIL e n/a 3 75 76 77 N B AE T e n/a 3 78 79 80 N B AX AE b n/a 3 81 82 83 N B B AE b n/a 3 84 85 86 N B SIL AE b n/a 3 87 88 89 N B SIL AE b n/a 3 90 91 92 N T AE AX e n/a 4 93 94 95 N T AE B e n/a 4 96 97 98 N T AE SIL e n/a 4 99 100 101 N T AE T e n/a 4 102 103 104 N T AX AE b n/a 4 105 106 107 N T B AE b n/a 4 108 109 110 N T SIL AE b n/a 4 111 112 113 N T T AE b n/a 4 114 115 116 N

is used as the base to give the following CD-tied model definition file with 39 tied states (senones):

    # triphone: (null) # seno map: (null) # 0.3 5 n_base 34 n_tri 156 n_state_map 54 n_tied_state 15 n_tied_ci_state 5 n_tied_tmat # # Columns definitions #base lft rt p attrib tmat ... state id's ... SIL - - - filler 0 0 1 2 N AE - - - n/a 1 3 4 5 N AX - - - n/a 2 6 7 8 N B - - - n/a 3 9 10 11 N T - - - n/a 4 12 13 14 N AE B T i n/a 1 15 16 17 N AE T B i n/a 1 18 16 19 N AX AX AX s n/a 2 20 21 22 N AX AX B s n/a 2 23 21 22 N AX AX SIL s n/a 2 358
24 21 22 N AX AX T s n/a 2 25 21 22 N AX B AX s n/a 2 26 21 27 N AX B B s n/a 2 23 21 27 N AX B SIL s n/a 2 24 21 27 N AX B T s n/a 2 25 21 27 N AX SIL AX s n/a 2 26 21 28 N AX SIL B s n/a 2 23 21 28 N AX SIL SIL s n/a 2 24 21 28 N AX SIL T s n/a 2 25 21 28 N AX T AX s n/a 2 26 21 29 N AX T B s n/a 2 23 21 29 N AX T SIL s n/a 2 24 21 29 N AX T T s n/a 2 25 21 29 N B AE AX e n/a 3 30 31 32 N B AE B e n/a 3 33 31 32 N B AE SIL e n/a 3 34 31 32 N B AE T e n/a 3 35 31 32 N B AX AE b n/a 3 36 37 38 N B B AE b n/a 3 36 37 39 N B SIL AE b n/a 3 36 37 40 N B T AE b n/a 3 36 37 41 N T AE AX e n/a 4 42 43 44 N T AE B e n/a 4 45 43 44 N T AE SIL e n/a 4 46 43 44 N T AE T e n/a 4 47 43 44 N T AX AE b n/a 4 48 49 50 N T B AE b n/a 4 48 49 51 N T SIL AE b n/a 4 48 49 52 N T T AE b n/a 4 48 49 53 N
Command-line Argument Description

Usage: [options]

-**help**  Shows the usage of the tool (Default Value : no)

-**example**  Shows example of how to use the tool (Default Value : no)

-**imoddeffn**  Untied-state model definition file (Default Value : NONE)

-**omoddeffn**  Tied-state model definition file (Default Value : NONE)

-**treedir**  SPHINX-III tree directory containing pruned trees (Default Value : NONE)

-**psetfn**  Phone set definiton file (Default Value : NONE)
A.2.34  wave2feat

**Tool Description**

Description:

Create cepstra from audio file.

The main parameters that affect the final output are srate, lowerf, upperf, nfilt, and nfft. Typical values for nfft are 256 and 512. The input format can be raw audio (pcm, two byte signed integer), NIST Sphere (.sph) or MS Wav (.wav).

Typical values are shown on table A.2.34.

<table>
<thead>
<tr>
<th>srate</th>
<th>8000</th>
<th>11025</th>
<th>16000</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowerf</td>
<td>200</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>upperf</td>
<td>3700</td>
<td>5200</td>
<td>16000</td>
</tr>
<tr>
<td>nfilt</td>
<td>31</td>
<td>37</td>
<td>40</td>
</tr>
</tbody>
</table>

**Example**

Example:

This example creates a cepstral file named "output.mfc" from an input audio file named "input.raw", which is a raw audio file (no header information), which was originally sampled at 16kHz.

```
wave2feat
   -i         input.raw
   -o         output.mfc
   -raw       no
   -input_endian little
   -samprate  16000
   -lowerf    130
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-upperf</td>
<td>6800</td>
</tr>
<tr>
<td>-nfilt</td>
<td>40</td>
</tr>
<tr>
<td>-nfft</td>
<td>512</td>
</tr>
</tbody>
</table>
Command-line Argument Description

Usage: [options]

- **-help** (Default Value: no)
  Description: Shows the usage of the tool

- **-example** (Default Value: no)
  Description: Shows example of how to use the tool

- **-i** (Default Value: NO DEFAULT VALUE)
  Description: Single audio input file

- **-o** (Default Value: NO DEFAULT VALUE)
  Description: Single cepstral output file

- **-c** (Default Value: NO DEFAULT VALUE)
  Description: Control file for batch processing

- **-di** (Default Value: NO DEFAULT VALUE)
  Description: Input directory, input file names are relative to this, if defined

- **-ei** (Default Value: NO DEFAULT VALUE)
  Description: Input extension to be applied to all input files

- **-do** (Default Value: NO DEFAULT VALUE)
  Description: Output directory, output files are relative to this

- **-eo** (Default Value: NO DEFAULT VALUE)
  Description: Output extension to be applied to all output files

- **-nist** (Default Value: no)
  Description: Defines input format as NIST sphere

- **-raw** (Default Value: no)
  Description: Defines input format as raw binary data
• **-mswav** (Default Value : no)
  Description: Defines input format as Microsoft Wav (RIFF)

• **-input_endian** (Default Value : little)
  Description: Endianness of input data, big or little, ignored if NIST or MS Wav

• **-nchans** (Default Value : 1)
  Description: Number of channels of data (interlaced samples assumed)

• **-whichchan** (Default Value : 1)
  Description: Channel to process

• **-logspec** (Default Value : no)
  Description: Write out logspectral files instead of cepstra

• **-feat** (Default Value : sphinx)
  Description: SPHINX format - big endian

• **-mach_endian** (Default Value : little)
  Description: Endianness of machine, big or little

• **-alpha** (Default Value : 0.97)
  Description: Preemphasis parameter

• **-srate** (Default Value : 16000.0)
  Description: Sampling rate

• **-frate** (Default Value : 100)
  Description: Frame rate

• **-wlen** (Default Value : 0.025625)
  Description: Hamming window length
• **-nfft** (Default Value : 512)
  Description: Size of FFT

• **-nfilt** (Default Value : 40)
  Description: Number of filter banks

• **-lowerf** (Default Value : 133.33334)
  Description: Lower edge of filters

• **-upperf** (Default Value : 6855.4976)
  Description: Upper edge of filters

• **-ncep** (Default Value : 13)
  Description: Number of cep coefficients

• **-doublebw** (Default Value : no)
  Description: Use double bandwidth filters (same center freq)

• **-blocksize** (Default Value : 200000)
  Description: Block size, used to limit the number of samples used at a time when reading very large audio files

• **-dither** (Default Value : no)
  Description: Add 1/2-bit noise

• **-verbose** (Default Value : no)
  Description: Show input filenames
A.2.35 QUICK_COUNT

Tool Description

Description:
Generate a set of context-dependent phones list (usually triphones) given a dictionary.

A phone list is first created in the following format:

phone1 0 0 0 0 phone2 0 0 0 0 phone3 0 0 0 0

The phone list of CI model training must be used to generate this.

Next a temporary dictionary is generated, which has all words except the filler word (word enclosed in ++()++). The entry. SIL SIL must be added to the temporary dictionary and the dictionary must be sort in alphabetic order.

-q: a mandatory flag that tell QUICK_COUNT to consider all word pairs while constructing the phone list -p: the formatted phone list -b: a temporary dictionary file -o: output triphone list

Example

Example:

QUICK_COUNT -q yes -p phonelist -b dictionary -o outputlist
Command-line Argument Description

Usage: [options]

-help Shows the usage of the tool (Default Value : no)
-examplE Shows example of how to use the tool (Default Value : no)
-v Verbose (Default Value : no)
-q Flag to consider all word pairs (Default Value : yes)
-b Base file or the dictionary file (Default Value : NONE)
-p Phone list file (Default Value : NONE)
-P Another argument for phone list file. Equivalent to -p. (keep it for backward compatibility purpose.) (Default Value : NONE)
-o Output triphone list (Default Value : NONE)
-i Input control file (Default Value : NONE)
-I Another argument for input control file. Equivalent to -i. (Keep it for backward compatibility purpose) (Default Value : NONE)
-f Find examples (?) (Default Value : no)
-S Use single path to make triphones (Default Value : no)
-s Directory of sentences (Default Value : NONE)
A.3 SphinxTrain: Perl Training Scripts

A.3.1 00.verify/verify_all.pl

A.3.2 01.vector_quantize/slave.VQ.pl

A.3.3 02.ci_schmm/slave_convg.pl

A.3.4 03.makeuntiedmdef/make_untied_mdef.pl

A.3.5 04.cd_schmm_untied/slave_convg.pl

A.3.6 05.buildtrees/slave.treebuilder.pl

A.3.7 06.prunetree/prunetree.pl

A.3.8 07.cd-schmm/slave_convg.pl

A.3.9 08.deleted-interpolation/deleted_interpolation.pl

A.3.10 Summary of Configuration Parameter in script level

A.3.11 Notes on parallelization
A.4 CMU LM Toolkit

A.4.1 binlm2arpa

Description

**binlm2arpa** : Convert a binary format language model to ARPA format.

**Input** : A binary format language model, as generated by idngram2lm.

**Output** : An ARPA format language model.

**Command Line Syntax:**

```
binlm2arpa -binary .binlm -arpa .arpa [ -verbosity 2 ]
```
A.4.2 evallm

**Input** : A binary or ARPA format language model, as generated by idngram2lm. In addition, one may also specify a text stream to be used to compute the perplexity of the language model. The ARPA format language model does not contain information as to which words are context cues, so if an ARPA format language model is used, then a context cues file may be specified as well.

**Output** : The program can run in one of two modes.

- **compute-PP** Output is the perplexity of the language model with respect to the input text stream.

- **validate** Output is confirmation or denial that the sum of the probabilities of each of the words in the context supplied by the user sums to one.

**Command Line Syntax:**

`evallm [ -binary .binlm — -arpa .arpa [ -context .ccs ] ]`

**Notes:** evallm can receive and process commands interactively. When it is run, it loads the language model specified at the command line, and waits for instructions from the user. The user may specify one of the following commands:

- **perplexity**

  Computes the perplexity of a given text. May optionally specify words from which to force back-off.

  Syntax:

  `perplexity -text .text [ -probs .fprobs ] [ -oovs .oov_file ] [ -annotate .annotation_file ] [ -backoff_from_unk_inc — -backoff_from_unk_exc ] [ -backoff_from_ccs_inc — -backoff_from_ccs_exc ] [ -backoff_from_list .fblist ] [ -include_unks ]`

  If the -probs parameter is specified, then each individual word probability will be written out to the specified probability stream file.

  If the -oovs parameter is specified, then any out-of-vocabulary (OOV) words which are encountered in the test set will be written out to the specified file.

  If the -annotate parameter is used, then an annotation file will be created, containing information on the probability of each word in the test set according to the language model, as well as the back-off class
for each event. The back-off classes can be interpreted as follows: Assume we have a trigram language model, and are trying to predict \( P(C \mid A, B) \). Then back-off class "3" means that the trigram "A B C" is contained in the model, and the probability was predicted based on that trigram. "3-2" and "3x2" mean that the model backed-off and predicted the probability based on the bigram "B C"; "3-2" means that the context "A B" was found (so a back-off weight was applied), "3x2" means that the context "A B" was not found.

To force back-off from all unknown words, use the `backoff_from_unk_inc` or `backoff_from_unk_exc` flag (the difference being the difference between inclusive or exclusive forced back-off). To force back-off from all context-cues, use the `backoff_from_ccs_inc` or `backoff_from_ccs_exc` flag. One can also specify a list of words from which to back-off, by storing this list in a forced back-off list file and using the `backoff_from_list` switch.

`-include_unks` results in a perplexity calculation in which the probability estimates for the unknown word are included.

- **validate**

  Calculate the sum of the probabilities of all the words in the vocabulary given the context specified by the user.

  Syntax:

  ```
  validate [ -backoff_from_unk_inc — -backoff_from_unk_exc ] [ -backoff_from_ccs_inc — -backoff_from_ccs_exc ] [ -backoff_from_list .fb-list ] word1 word2 ... word_(n-1)
  ```

  Where n is the n in n-gram.

- **help**

  Displays a help message.

  Syntax: `help`

- **quit**

  Exits the program.

  Syntax:

  `quit`

Since the commands are read from standard input, a command file can be piped into it directly, thus removing the need for the program to run interactively:

```bash
$ echo "perplexity -text b.text" | evallm -binary a.binlm
```
A.4.3 idngram2lm

**Input**: An id n-gram file (in either binary (by default) or ASCII (if specified) mode).

**Output**: A list of the frequency-of-frequencies for each of the 2-grams, ..., n-grams, which can enable the user to choose appropriate cut-offs, and to specify appropriate memory requirements with the `-spec_num` option in idngram2lm.

**Command Line Syntax:**

```
idngram2lm -idngram .idngram -vocab .vocab -arpa .arpa [-binary .binlm] [ -context .ccs ] [ -calc_mem — -buffer 100 — -spec_num y ... z ] [ -vocab_type 1 ] [ -oov_fraction 0.5 ] [ -two_byte_bo_weights [ -min_bo_weight nnnnn] [ -max_bo_weight nnnnn] [ -out_of_range_bo_weights] ] [ -four_byte_counts ] [ -linear — -absolute — -good_turing — -witten_bell ] [ -disc_ranges 1 7 7 ] [ -cutoffs 0 ... 0 ] [ -min_unicount 0 ] [ -zeroton_fraction ] [ -ascii_input — -bin_input ] [ -n 3 ] [ -verbosity 2 ]
```

**Usage:**

The `-context` parameter allows the user to specify a file containing a list of words within the vocabulary which will serve as context cues (for example, markers which indicate the beginnings of sentences and paragraphs).

- `-calc_mem`, `-buffer` and `-spec_num x y ... z` are options to dictate how it is decided how much memory should be allocated for the n-gram counts data structure. `-calc_mem` demands that the id n-gram file should be read twice, so that we can accurately calculate the amount of memory required. `-buffer` allows the user to specify an amount of memory to grab, and divides this memory equally between the 2,3, ..., n-gram tables. `-spec_num` allows the user to specify exactly how many 2-grams, 3-grams, ..., and n-grams will need to be stored. The default is `-buffer STD_MEM`.

The toolkit provides for three types of vocabulary, which each handle out-of-vocabulary (OOV) words in different ways, and which are specified using the `-vocab_type` flag.

A closed vocabulary (-`vocab_type 0`) model does not make any provision for OOVs. Any such words which appear in either the training or test data will cause an error. This type of model might be used in a command/control environment where the vocabulary is restricted to the number of commands that the system understands, and we can therefore guarantee that no OOVs will occur in the training or test data.

An open vocabulary model allows for OOVs to occur; out of vocabulary words are all mapped to the same symbol. Two types of open vocabulary model are implemented in the toolkit. The first type (-`vocab_type 1`) treats
this symbol the same way as any other word in the vocabulary. The second

type (-vocab_type 2) of open vocabulary model is to cover situations where

no OOVs occurred in the training data, but we wish to allow for the situ-

ation where they could occur in the test data. This situation could occur,

for example, if we have a limited amount of training data, and we choose a

vocabulary which provides a proportion of the discount probability mass

(specified by the -oov_fraction option) is reserved for OOV words.

The discounting strategy and its parameters are specified by the -linear,

-absolute, -good_turing and -witten_bell options. With Good Turing dis-

counting, one can also specify the range over which discounting occurs,

using the -disc_ranges option.

The user can specify the cutoffs for the 2-grams, 3-grams, ..., n-grams

by using the -cutoffs parameter. A cutoff of K means that all n-grams oc-

curring K or fewer times are discarded. If the parameter is omitted, then

all the cutoffs are set to zero. The -zero_fraction option specifies that

P(zeroton) (the unigram probability assigned to a vocabulary word that did

not occurred at all in the training data) will be at least that fraction of

P(singleton) (the probability assigned to a vocabulary word that occurred

exactly once in the training data).

By default, the n-gram counts are stored in two bytes by use of a

count table (this allows the counts to exceed 65535, while keeping the
data structures used to store the model compact). However, if more than
65535 distinct counts need to be stored (very unlikely, unless construct-
ing 4-gram or higher language models using Good-Turing discounting), the
-four_byte_counts option will need to be used.

The floating point values of the back-off weights may be stored as two-
byte integers, by using the -two_byte_alphas switch. This will introduce
slight rounding errors, and so should only be used if memory is short. The
-min_alpha, -max_alpha and -out_of_range_alphas are parameters used
by the functions for using two-byte alphas. Their values should only be
altered if the program instructs it. For further details, see the comments
in the source file src/two_byte_alphas
A.4.4  idngram2stats

**indngram2stats** : Report statistics for an id n-gram file.

**Input** : An id n-gram file (in either binary (by default) or ASCII (if specified) mode).

**Output** : A list of the frequency-of-frequencies for each of the 2-grams, ..., n-grams, which can enable the user to choose appropriate cut-offs, and to specify appropriate memory requirements with the **-spec_num** option in idngram2lm.

**Command Line Syntax:**

```
indngram2stats [ -n 3 ] [ -fof_size 50 ] [ -verbosity 2 ] [ -ascii_input ] < .idngram > .stats
```
A.4.5 interpolate

**Input**: Files containing probability streams, as generated by the -probs option of the perplexity command of evallm. Alternatively these probabilities could be generated from a separate piece of code, which assigns word probabilities according to some other language model, for example a cache-based LM. This probability stream can then be linearly interpolated with one from a standard n-gram model using this tool.

**Output**: An optimal set of interpolation weights for these probability streams, and (optionally) a probability stream corresponding to the linear combination of all the input streams, according to the optimal weights. The optimal weights are calculated using the expectation maximisation (EM) algorithm.

Command Line Syntax:

```
interpolate +[-] model1.fprobs +[-] model2.fprobs ... [ -test_all — -test_first n — -test_last n — -cv ] [ -tag .tags ] [ -captions .captions ] [ -in_lambdas .lambdas ] [ -out_lambdas .lambdas ] [ -stop_ratio 0.999 ] [ -probs .fprobs ] [ -max_probs 6000000 ]
```

The probability stream filenames are prefaced with a + (or a +- to indicate that the weighting of that model should be fixed).

There are a range of options to determine which part of the data is used to calculate the weights, and which is used to test them. One can test the perplexity of the interpolated model based on all the data, using the -test_all option, in which case a set of lambdas must also be specified with the -lambda option (i.e. the lambdas are pre-specified, and not calculated by the program). One can specify that the first or last n items are the test set by use of the -test_first n or -test_last n options. Or one can perform two-way cross validation using the -cv option. If none of these are specified then the whole of the data is used for weight estimation.

By default, the initial interpolation weights are fixed as 1/number_of_models, but alternative values can be stored in a file and used via the -in_lambdas option.

The -probs switch allows the user to specify a filename in which to store the combined probability stream. The optimal lambdas can also be stored in a file by use of the -out_lambdas command.

The program stops when the ratio of the test-set perplexity between two successive iterations is above the value specified in the -stop_ratio option.

The data can be partitioned into different classes (with optimisation being performed separately on each class) using the -tags parameter. The tags file will contain an integer for each item in the probability streams.
corresponding to the class that the item belongs to. A file specified using
the -captions option will allow the user to attach names to each of the
classes. There should be one line in the captions file for each tag, with
each line corresponding to the name of the tag.

The amount of memory allocated to store the probability streams is
dictated by the -max_probs option, which indicates the maximum number
of probabilities allowed in one stream.
A.4.6 mergeidngram

Input: A set of id n-gram files (in either binary (by default) or ASCII (if specified) format - note that they should all be in the same format, however).

Output: One id n-gram file (in either binary (by default) or ASCII (if specified) format), containing the merged id n-grams from the input files.

Notes: This utility can also be used to convert id n-gram files between ascii and binary formats.

Command Line Syntax:

mergeidngram [ -n 3 ] [ -ascii_input ] [ -ascii_output ] .idngram.1 .idngram.2 ... .idngram.N ¿ .idngram

A.4.7 ngram2mgram

Input: Either a word n-gram file, or an id n-gram file.

Output: Either a word m-gram file, or an id m-gram file, where m ¡ n.

Command Line Syntax:

ngram2mgram -n N -m M [ -binary — -ascii — -words ] < .ngram > .mgram

The -binary, -ascii, -words correspond to the format of the input and output (Note that the output file will be in the same format as the input file). -ascii and -binary denote id n-gram files, in ASCII and binary formats respectively, and -words denotes a word n-gram file.
A.4.8 text2idngram

**Input**: Text stream, plus a vocabulary file.

**Output**: List of every id n-gram which occurred in the text, along with its number of occurrences.

**Notes**: Maps each word in the text stream to a short integer as soon as it has been read, thus enabling more n-grams to be stored and sorted in memory.

Command Line Syntax:

```
text2idngram -vocab .vocab [ -buffer 100 ] [ -temp /usr/tmp/ ] [ -files 20 ] [ -gzip — -compress ] [ -n 3 ] [ -write_ascii ] [ -fof_size 10 ] [ -verbosity 2 ] < .text > .idngram
```

By default, the id n-gram file is written out as binary file, unless the **-write_ascii** switch is used.

The size of the buffer which is used to store the n-grams can be specified using the **-buffer** parameter. This value is in megabytes, and the default value can be changed from 100 by changing the value of STD_MEM in the file src/toolkit.h before compiling the toolkit.

The program will also report the frequency of frequency of n-grams, and the corresponding recommended value for the **-spec_num** parameters of idngram2lm. The **-fof_size** parameter allows the user to specify the length of this list. A value of 0 will result in no list being displayed.

The **-temp** option allows the user to specify where the program should store its temporary files.

In the case of really huge quantities of data, it may be the case that more temporary files are generated than can be opened at one time by the filing system. In this case, the temporary files will be merged in chunks, and the **-files** parameter can be used to specify how many files are allowed to be open at one time.
A.4.9  text2wfreq

**Input**: Text stream

**Output**: List of every word which occurred in the text, along with its number of occurrences.

**Notes**: Uses a hash-table to provide an efficient method of counting word occurrences. Output list is not sorted (due to “randomness” of the hash-table), but can be easily sorted into the user’s desired order by the UNIX sort command. In any case, the output does not need to be sorted in order to serve as input for wfreq2vocab.

Usage: text2wfreq [ -hash 1000000 ] [ -verbosity 2 ] < .text > .wfreq

Higher values for the -hash parameter require more memory, but can reduce computation time.
A.4.10  text2wngram

**Input**: Text stream

**Output**: List of every word n-gram which occurred in the text, along with its number of occurrences.

The maximum numbers of characters and words that can be stored in the buffer are given by the -chars and -words options. The default number of characters and words are chosen so that the memory requirement of the program is approximately that of STD_MEM, and the number of characters is seven times greater than the number of words.

The -temp option allows the user to specify where the program should store its temporary files.

**Usage**: text2wngram [ -n 3 ] [ -temp /usr/tmp/ ] [ -chars 63636363 ] [ -words 9090909 ] [ -gzip — -compress ] [ -verbosity 2 ] < .text > .wngram
A.4.11 wfreq2vocab

Example

wfreq2vocab : Generate a vocabulary file from a word frequency file.

**Input** : A word unigram file, as produced by text2wfreq

**Output** : A vocabulary file.

The `-top` parameter allows the user to specify the size of the vocabulary; if the program is called with the command `-top 20000`, then the vocabulary will consist of the most common 20,000 words.

The `-gt` parameter allows the user to specify the number of times that a word must occur to be included in the vocabulary; if the program is called with the command `-gt 10`, then the vocabulary will consist of all the words which occurred more than 10 times.

If neither the `-gt`, nor the `-top` parameters are specified, then the program runs with the default setting of taking the top 20,000 words.

The `-records` parameter allows the user to specify how many of the word and count records to allocate memory for. If the number of words in the input exceeds this number, then the program will fail, but a high number will obviously result in a higher memory requirement.

**Usage** : wfreq2vocab [ -top 20000 — -gt 10] [ -records 3000000 ] [ -verbosity 2] < .wfreq > .vocab
A.4.12 wngram2idngram

**Input**: Word n-gram file, plus a vocabulary file.

**Output**: List of every id n-gram which occurred in the text, along with its number of occurrences, in either ASCII or binary format.

**Note**: For this program to be successful, it is important that the vocabulary file is in alphabetical order. If you are using vocabularies generated by the wfreq2vocab tool then this should not be an issue, as they will already be alphabetically sorted.

**Command Line Syntax**:

```
wngram2idngram -vocab .vocab [ -buffer 100 ] [ -hash 200000 ] [ -temp /usr/tmp/ ] [ -files 20 ] [ -gzip — -compress ] [ -verbosity 2 ] [ -n 3 ] [ -write_ascii ] < .wngram > .idngram
```

The size of the buffer which is used to store the n-grams can be specified using the `-buffer` parameter. This value is in megabytes, and the default value can be changed from 100 by changing the value of STD_MEM in the file src/toolkit.h before compiling the toolkit.

The program will also report the frequency of n-grams, and the corresponding recommended value for the `-spec_num` parameters of idngram2lm. The `-fof_size` parameter allows the user to specify the length of this list. A value of 0 will result in no list being displayed.

Higher values for the `-hash` parameter require more memory, but can reduce computation time.

The `-temp` option allows the user to specify where the program should store its temporary files.

The `-files` parameter is used to specify the number of files which can be open at one time.
Appendix B

Frequently Asked Questions

B.1 Introduction

Author: Arthur Chan, Editor: Arthur Chan

[Editor Notes: Directly converted from Dr. Rita Singh’s FAQ page and the FAQ page from cmusphinx.org]

For a very long time, Dr. Rita Singh was in robust speech recognition group of the speech group of CMU. She has a lot of real-life experience in using the Sphinx 2, Sphinx 3 and Sphinx 4’s recognizer. Many beginners of speech recognition was benefited by her web page. The following is a set of FAQ she collected in the 7 years she was in CMU. Evandro Gouvea has become the maintainer of the page and augmented a lot to this list.

B.2 Data-preparation for training acoustic models.

B.3 Telling size of speech corpus in hours

Question: How can I tell the size of my speech corpus in hours? Can I use it all for training?
Answer: You can only train with utterances for which you have transcripts. You cannot usually tell the size of your corpus from the number of utterances you have. Sometimes utterances are very long, and at other times they may be as short as a single word or sound. The best way to estimate the size of your corpus in hours is to look at the total size in bytes of all utterance files which you can use to train your models. Speech data are usually stored in integer format. Assuming that is so and ignoring any headers that your file might have, an approximate estimate of the size of your corpus in hours can be obtained from the following parameters of the speech data:

- Sampling Rate: If this is S KiloHertz, then there are S x1000 samples or integers in every second of your data.
- Sample Size: If your sampling is "8bit" then every integer has 1 byte associated with it. If it is "16bit" then every integer in your data has 2 bytes associated with it.
- Hour Size: 3600 seconds in an hour

Here's a quick reference table:

<table>
<thead>
<tr>
<th>No. of bytes</th>
<th>Sampling rate</th>
<th>Sample size</th>
<th>Hours of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>8khz</td>
<td>8bit</td>
<td>X / (8000<em>1</em>3600)</td>
</tr>
<tr>
<td>X</td>
<td>16khz</td>
<td>16bit</td>
<td>X / (1600<em>2</em>3600)</td>
</tr>
</tbody>
</table>

B.4 Force alignment

Question: What is force-alignment? Should I force-align my transcripts before I train?

Answer: The process of force-alignment takes an existing transcript, and finds out which, among the many pronunciations for the words occurring in the transcript, are the correct pronunciations. So when you refer to "force-aligned" transcripts, you are also inevitably referring to a *dictionary* with reference to which the transcripts have been force-aligned. So if you have two dictionaries and one has the word “PANDA” listed as:

PANDA P AA N D AA PANDA(2) P AE N D AA PANDA(3) P AA N D AX

and the other one has the same word listed as

PANDA P AE N D AA PANDA(2) P AA N D AX PANDA(3) P AA N D AA

And you force-aligned using the first dictionary and get your transcript to look like:
I SAW A PANDA(3) BEAR,

then if you used that transcript to train but used the second dictionary
to train, then you would be giving the wrong pronunciation to the trainer.
You would be telling the trainer that the pronunciation for the word PANDA
in your corpus is "P AA N D AA" instead of the correct one, which should
have been "P AA N D AX". The data corresponding to the phone AX will now
be wrongly used to train the phone AA.

What you must really do is to collect your transcripts, use only the
first listed pronunciation in your training dictionary, train ci models, and
use *those ci models* to force-align your transcripts against the training
dictionary. Then go all the way back and re-train your ci models with the
new transcripts.

**Question:** I have about 7000 utterances or 12 hours of speech in my
training set. I found fa ligned transcripts for all but 114 utterances, and
those 114 utterances have no transcripts that I can find. Should I leave
them out of the training? I don’t think it will make that much difference at
its 1.4 data, how many senones should I use?

**Answer:** Leave out utterances for which you don’t have transcripts (un-
less you have very little data in the first place, in which case hear out the
audio and transcribe it yourself). In this case, just leave them out.

**Augmented by Arthur at 2005015:** It was the intention of the de-
signer to disallow some utterance to be force-aligned by align. For one
example, if there is off-by-one error of the input control file. Failure of
align will give a warning to the user such that they will detect this prob-
lem early.

**Question:** I don’t have transcripts. How can I force-align?

**Answer:** you cannot force-align any transcript that you do not have.

**Augmented by Arthur at 2005015:** One thing you could do if you don’t
have a transcribe is to use one model to do speech recognition and assume
the answer is the transcript. In this way, you will be able to get end-points
of words using align. Obviously If you do in this way, the quality of the
end-points will depend on the recognition results.

**Question:** I am going to first train a set of coarse models to force-
align the transcripts. So I should submit beginning and end silence marked
transcripts to the trainer for the coarse models. Currently I am keeping all
the fillers, such as UM, BREATH, NOISE etc. in my transcripts, but wrapped
with "+". Do you think the trainer will consider them as fillers instead of
normal words?

**Answer:** According to the trainer, ANY word listed in the dictionary in
terms of any phone/sequence of phones is a valid word. BUT the decision
tree builder ignores any +word+ phone as a noise phone and does not build decision trees for the phone. So while training, mark the fillers as ++anything++ in the transcript and then see that either the filler dictionary or the main dictionary has some mapping.

++anything++ +something+

where +something+ is a phone listed in your phonelist.

**Question:** I have a huge collection of data recorded under different conditions. I would like to train good speaker-independent models using this (or a subset of this) data. How should I select my data? I also suspect that some of the transcriptions are not very accurate, but I can’t figure out which ones are inaccurate without listening to all the data.

**Answer:** If the broad acoustic conditions are similar (for example, if all your data has been recorded off TV shows), it is best to use all data you can get for training speaker-independent bandwidth-independent models, gender-independent models. If you suspect that some of the data you are using might be bad for some reason, then during the baum-welch iterations you can monitor the likelihoods corresponding to each utterance and discard the really low-likelihood utterances. This would filter out the bad acoustic/badly transcribed data.

**Question:** What is the purpose of the 4th field in the control file:

**Answer:** The fourth field in the control file is simply an utterance identifier. So long as that field and the entry at the end of the corresponding utterance in the transcript file are the same, you can have anything written there and the training will go through. It is only a very convenient tag. The particular format that you see for the fourth field is just an “informative” way of tagging. Usually we use file paths and names along with other file attributes that are of interest to us.

**Data conversion**

**Question:** I am trying to train with Switchboard data. Switchboard data is mulaw encoded. Do we have generic tools for converting from stereo mulaw to standard raw file?

**Answer:** NIST provides a tool called w_edit which lets you specify the output format, the desired channel to decode and the beginning and ending sample that you would like decoded. ch_wave, a part of the Edinburgh speech tools, does this decoding as well (send mail to awb@cs.cmu.edu for more information on this). Here is a conversion table for converting 8 bit mulaw to 16 bit PCM. The usage must be clear from the table - linear_value

<table>
<thead>
<tr>
<th>Linear Value</th>
<th>8 bit Mulaw</th>
<th>16 bit PCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>386</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
= linear[mu_law_value]; (i.e. if your mu law value is 16, the PCM value is linear[16]);
--- mu-law to PCM conversion table ---

static short int linear[256] = -32124, -31100, -30076, -29052, -28028, -
27004, -25980, -24956, -23932, -22908, -21884, -20860, -19836, -18812, -
17788, -16764, -15740, -14716, -13692, -12668, -11644, -10620, -9596, -8572, -
7548, -6524, -5500, -4476, -3452, -2428, -1404, -384, 388,

--- mu-law to PCM conversion table ---
B.5 Selecting modeling parameters.

B.6 Rule of thumb of choosing parameters

**Question:** How many senones should I train?

**Answer:** Thumb rule figures for the number of senones that you should be training are given in the following table:

<table>
<thead>
<tr>
<th>Amount of training data(hours)</th>
<th>No. of senones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>500-1000</td>
</tr>
<tr>
<td>4-6</td>
<td>1000-2500</td>
</tr>
<tr>
<td>6-8</td>
<td>2500-4000</td>
</tr>
<tr>
<td>8-10</td>
<td>4000-5000</td>
</tr>
<tr>
<td>10-30</td>
<td>5000-5500</td>
</tr>
<tr>
<td>30-60</td>
<td>5500-6000</td>
</tr>
<tr>
<td>60-100</td>
<td>6000-8000</td>
</tr>
<tr>
<td>Greater than 100</td>
<td>8000 are enough</td>
</tr>
</tbody>
</table>

**Question:** How many states-per-hmm should I specify for my training?

**Answer:** If you have "difficult" speech (noisy/spontaneous/damaged), use 3-state hmm's with a noskip topology. For clean speech you may choose to use any odd number of states, depending on the amount of data you have and the type of acoustic units you are training. If you are training word models, for example, you might be better off using 5 states or higher. 3-5 states are good for shorter acoustic units like phones. You cannot currently train 1 state hmm's with the Sphinx.

Remember that the topology is also related to the frame rate and the minimum expected duration of your basic sound units. For example the phoneme "T" rarely lasts more than 10-15 ms. If your frame rate is 100 frames per second, "T" will therefore be represented in no more than 3 frames. If you use a 5 state noskip topology, this would force the recognizer to use at least 5 frames to model the phone. Even a 7 state topology that permits skips between alternate states would force the recognizer to visit at least 4 of these states, thereby requiring the phone to be at least 4 frames long. Both would be erroneous. Give this point very serious thought before you decide on your HMM topology. If you are not convinced, send us a mail and we'll help you out.

**Question:** I have two sets of models, A and B. The set A has been trained with 10,000 tied states (or senones) and B has been trained with 5,000 senones. If I want to compare the recognition results on a third database
using A and B, does this difference in the number of senones matter?

**Answer:** If A and B have been optimally trained (i.e. the amount of data available for training each has been well considered), then the difference in the number of tied states used should not matter.
B.7 Feature computation

FEATURE COMPUTATION

**Question:** How appropriate are the standard frame specifications for feature computation? I am using the default values but the features look a bit "shifted" with respect to the speech waveform. Is this a bug?

**Answer:** There are two factors here: the frame *size* and the frame *rate*. Analysis frame size is typically 25 ms. Frame rate is 100 frames/sec. In other words, we get one frame every 10 ms (a nice round number), but we may need to adjust boundaries a little bit because of the frame size (a 5ms event can get smeared over three frames - it could occur in the tail end of one frame, the middle of the next one, and the beginning of the third, for the 10ms frame shifts). The feature vectors sometimes look shifted with respect to the speech samples. However, there is no shift between the frames and the speech data. Any apparent shift is due to smearing. We do frequently get an additional frame at the end of the utterance because we pad zeros, if necessary, after the final samples in order to fill up the final frame.

**Question:** How do I find the center frequencies of the Mel filters?

**Answer:** The mel function we use to find the mel frequency for any frequency x is

\[(2595.0 \times (\text{float32})\text{log10}(1.0 + x/700.0))\]

substitute x with the upper and lower frequencies, subtract the results, and divide by the number of filters you have + 1 : that will give you the bandwidth of each filter as twice the number you get after division. The number you get after division + the lower frequency is the center frequency of the first filter. The rest of the center frequencies can be found by using the bandwidths and the knowledge that the filters are equally spaced on the mel frequency axis and overlap by half the bandwidth. These center frequencies can be transformed back to normal frequency using the inverse mel function

\[(700.0 \times ((\text{float32})\text{pow}(10.0, x/2595.0) - 1.0))\]

where x is now the center frequency.

**Question:** Does the front-end executable compute difference features?

**Answer:** No. The difference features are computed during runtime by the SPHINX-III trainer and decoders.

**Question:** What would be the consequence of widening the analysis windows beyond 25ms for feature computation?

**Answer:** Analysis windows are currently 25 ms wide with 10ms shifts.
Widening them would have many undesirable consequences:

- spectra will get more smoothed, and so you'll lose information for recognition
- smaller phones would get completely obliterated
- deltas will no longer be so informative (remember that in the dropping- every-other-frame experiment they are computed before dropping), as the time lags considered for their computation will be larger
- We engineer the system to enable HMM states to capture steady state regions. When a single frame begins representing changing information (when it is long, it will capture multiple events within the same frame, and not represent any of them accurately. Speech is steady only in about 25ms sections), the states will no longer capture the kind of classification information we want them to. The models will result in poor recognition as a consequence.

**Question:** I ran the wave2feat program with the configuration of srate = 16000 and nfft = 256, then the program crashed. I changed the nfft to 512, it works. So I'd like to know why.

**Answer:** At a sampling rate of 16000 samples/sec, a 25ms frame has 400 samples. If you try to fill these into 256 locations of allocated memory (in the FFT) you will have a segmentation fault. There *could* have been a check for this in the FFT code, but the default for 16kHz has been set correctly to be 512, so this was considered unnecessary.
Modeling filled pauses and non-speech events.

**Question:** Can you explain the difference between putting the words as fillers ++()++ instead of just putting them in the normal dictionary? My dictionary currently contains pronunciations for UH-HUH, UH-HUH(2) and UH-HUH(3). Should all of these effectively be merged to ++UH-HUH++ and mapped to a single filler phone like +UH-HUH+?

**Answer:** Putting them as normal words in the dictionary should not matter if you are training CI models. However, at the CD stage when the list of training triphones is constructed, the phones corresponding to the (++ ++) entries are mapped by the trainer to silence. For example the triphone constructed from the utterance

++UM++ A ++AH++

would be AX(SIL,SIL) and not AX(+UM+,AA) [if you have mapped ++UM++ to +UM+ and ++AH++ to the phone AA for training, in one of the training dictionaries]

Also, when you put ++()++ in the main dictionary and map it to some sequence of phones other than a single +() phone, you cannot build a model for the filler. For example UH-HUH may be mapped to AH HH AX , AX HH AX etc in the main dict, and when you train, the instances of UH-HUH just contribute to the models for AH, AX or HH and the corresponding triphones. On the other hand, if you map ++UH-HUH++ to +UH-HUH+, you can have the instances contribute exclusively to the phone +UH-HUH+. The decision to keep the filler as a normal word in the training dictionary and assign alternate pronunciations to it OR to model it exclusively by a filler phone must be judiciously made keeping the requirements of your task in mind.

During decoding and in the language model, the filler words ++()++ are treated very differently from the other words. The scores associated are computed in a different manner, taking certain additional insertion penalties into account.

Also, the SPHINX-II decoder is incapable of using a new filler unless there is an exclusive model for it (this is not the case with the SPHINX-III decoder). If there isn’t, it will treat the filler as a normal dictionary word and will ignore it completely if it is not there in the language model (which usually doesn’t have fillers), causing a significant loss in accuracy for some tasks.

**Question:** My training data contains no filler words (lipsmack, cough
etc.) Do you think I should retrain trying to insert fillers during forced align-
ment so that I could train on them? Since what I have is spontaneous speech,
I can’t imagine that in all 20000 utterances there are no filled pauses etc.

**Answer** Don’t use falign to insert those fillers. The forced aligner has a
tendency to arbitrarily introduce fillers all over the place. My guess is that
you will lose about 5 fillers to model. If you are going to use the SPHINX-III
decoder, however, you can compose some important fillers like "UH" and
"UM" as "AX" or "AX HH" or "AX M" and use them in the fillerdict. However,
the sphinx-2 decoder cannot handle this. If possible, try listening to some
utterances and see if you can insert about 50 samples of each filler - that
should be enough to train them crudely.

**Question:** How is SIL different from the other fillers? Is there any
special reason why I should designate the filler phones as +0+? What if I
*want* to make filler triphones?

**Answer** Silence is special in that it forms contexts for triphones, but
doesn’t have it’s own triphones (for which it is the central phone, ie). The
fillers neither form contexts nor occur as independent triphones. If you
want to build triphones for a filler, then the filler must be designated as a
proper phone without the "+" in the dictionaries.

**Question:** What is the meaning of the two columns in the fillerdict? I
want to reduce the number of fillers in my training.

**Answer** In a filler dictionary, we map all non-speech like sounds to
some phones and we then train models for those phones. For example, we
may say ++GUNSHOT++ +GUNSHOT+

The meaning is the same as "the pronunciation of the word ++GUN-
SHOT++ in the transcripts must be interpreted to be +GUNSHOT++" Now
if I have five more filler words in my transcripts: ++FALLINGWATER++
++LAUGH++ ++BANG++ ++BOMBING++ ++RIFLESHOT++

Then I know that the sounds of ++BANG++, ++BOMBING++ and ++RI-
FLESHOT++ are somewhat similar, so I can reduce the number of filler
phones to be modelled by modifying the entries in the filler dict to look like

++GUNSHOT++ +GUNSHOT+
++BANG++ +GUNSHOT+
++BOMBING++ +GUNSHOT+
++RIFLESHOT++ +GUNSHOT+
++FALLINGWATER++ +WATERSOUND+
++LAUGH++ +LAUGHSOUND+

so we have to build models only for the phones +GUNSHOT+, +WATER-
SOUND+ and +LAUGHSOUND+ now.
B.9 Training speed.

**Question:** I am trying to train models on a single machine. I just want to train a set of coarse models for forced-alignment. The baum-welch iterations are very slow. In 24 hours, it has only gone through 800 utterances. I have total 16,000 utterances. As this speed, it will take 20 days for the first iteration of baum-welch, considering the convergence ratio to be 0.01, it will take several months to obtain the first CI-HMM, let alone CD-HMM. Is there any way to speed this up?

**Answer** If you start from flat-initialized models the first two iterations of baum-welch will always be very slow. This is because all paths through the utterance are similar and the algorithm has to consider all of them. In the higher iterations, when the various state distributions begin to differ from each other, the computation speeds up a great deal.

Given the observed speed of your machine, you cannot possibly hope to train your models on a single machine. You may think of assigning a lower value to the "topn" argument of the bw executable, but since you are training CI models, changing the topn value from its default (99) to any smaller number will not affect the speed, since there is only at best 1 Gaussian per state anyway throughout the computation.

Try to get more machines to share the jobs. There is a -npart option to help you partition your training data. Alternatively, you can shorten your training set, since you only want to use the models for forced alignment. Models trained with about 10 hours of data will do the job just as well.
B.10 **Questions specific to log files.**

**Question:** My decode log file gives the following message:

```
ERROR: "../feat.c", line 205: Header size field: -1466929664(a8906e00); filesize: 1(00000001) ================ exited with status 0
```

**Answer:** The feature files are byte swapped!

**Question:** During force-alignment, the log file has many messages which say "Final state not reached" and the corresponding transcripts do not get force-aligned. What’s wrong?

**Answer:** The message means that the utterance likelihood was very low, meaning in turn that the sequence of words in your transcript for the corresponding feature file given to the force-aligner is rather unlikely. The most common reasons are that you may have the wrong model settings or the transcripts being considered may be inaccurate. For more on this go to Viterbi-alignment

**Question:** I am trying to do flat-initialization for training ci models. The cp_parm program is complaining about the -feat option. The original script did not specify a -feat option, however the cp_parm program complained that the default option was unimplemented. I’ve made several attempts at specifying a -feat option with no luck. Below is the output of two run. Can you give me an idea of what is happening here?

Default (no -feat passed) produces:

```
ERROR: "../feat.c", line 121: Unimplemented feature
c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1]
c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1]
```

**Answer:** The features files are byte swapped!
gau 1 j= 0
gau 2 j= 0

This is the error message if I attempt to specify the -feat option:
-feat c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1]

....
ERROR: "./feat.c", line 121: Unimplemented feature
c[1..L-1]d[1..L-1]c[0]d[0]dd[0]dd[1..L-1]
ERROR: "./feat.c", line 122: Implemented features are:
c/1..L-1/,d/1..L-1/,c/0/d/0/dd/0/,dd/1..L-1/
c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/
c/0..L-1/d/0..L-1/dd/0..L-1/
c/0..L-1/d/0..L-1/

Answer The last three lines in the case when you do not specify the
-feat option say that the cp_parm is going through and the mean vector
labelled "0" is being copied to state 0, state 1, state 2.... The same "0"
vector is being copied because this is a flat initialization where all means,
variances etc are given equal flat values. At this point, these errors in the
log files can just be ignored.

Question: I am trying to make linguistic questions for state tying. The
program keeps failing because it can’t allocate enough memory. Our ma-
chines are rather large with 512MB and 1 to 2 GB swap space. Does it
make sense that it really doesn’t have enough memory, or is it more likely
something else failed? Below is the log from this program.

-varfn
path[/model_parameters/new_fe.ci_continuous/variances
-mixwfn
[(path[/model_parameters/new_fe.ci_continuous/mixture_weights
-npermute 168
-niter 0
-qstperstt 20

...... ...... ......
INFO: ../s3gau_io.c(128): Read
 /sphx_train/hub97/training/model_parameters/new_fe.ci_continuous/means
[153x1x1 array]
INFO: ../s3gau_io.c(128): Read
/sphx_train/hub97/training/model_parameters/new_fe.ci_continuous/variances
[153x1x1 array]
FATAL_ERROR: "../ckd_alloc.c", line 109: ckd_calloc_2d failed for caller at
../main.c(186) at ../ckd_alloc.c(110)

Answer makequests searches \(2^n\) permute combinations several times for the optimal clustering of states. For this, it has to store \(2^n\) permute values (for the comparison). So, setting -npermute to anything greater than 8 or 10 makes the program very slow, and anything over 28 will make the program fail. We usually use a value of 8.

Question: I’m getting a message about end of data beyond end of file from agg_seg during vector-quantization. I assume this means the .ctl file references a set of data beyond the end of the file. Should I ignore this?

Answer Yes, for agg_seg if its going through in spite of the message. Agg-seg only collects samples of feature vectors to use for quantization through kmeans. No, for the rest of the training because it may cause random problems. The entry in the control file and the corresponding transcript have to be removed, if you cannot correct them for some reason.
B.11 Vector-quantization for discrete and semi-continuous models

**Question:** I have a question about VQ. When you look at the 39-dimensional [cep + d-cep + dd-cep] vector, it’s clear that each part (cep, d-cep, dd-cep) will have quite a different dynamic range and different mean. How should we account for this when doing DISCRETE HMM modeling? Should we make a separate codebook for each? If so, how should we “recombine” when recognizing? Or should we rescale the d-cep and dd-cep up so they can “compete” with the “larger” cep numbers in contributing to the overall VQ?

**Question:** Continue: In other words, suppose we want to train a complete discrete HMM system - is there a way to incorporate the d-cep and dd-cep features into the system to take advantage of their added information? If we just concatenate them all into one long vector and do standard VQ, the d-cep and dd-cep won’t have much of an influence as to which VQ codebook entry matches best an incoming vector. Perhaps we need to scale up the d-cep and dd-cep features so they have the same dynamic range as the cep features? Is there a general strategy that people have done in the past to make this work? Or do we have to “bite the bullet” and move up to semi-continuous HMM modeling?

**Answer** You *could* add d-cep and dd-cep with the cepstra into one long feature. However, this is always inferior to modeling them as separate feature streams (unless you use codebooks with many thousand code-words).

Secondly, for any cepstral vector, the dynamic range and value of c[12], for example, is much smaller (by orders of magnitude) than c[1] and doesn’t affect the quantization at all. In fact, almost all the quantization is done on the basis of the first few cepstra with the largest dynamic ranges. This does not affect system performance in a big way. One of the reasons is that the classification information in the features that do not affect VQ much is also not too great.

However, if you do really want to be careful with dynamic ranges, you could perform VQ using Mahalanobis distances, instead of Euclidean distances. In the Mahalanobis distance each dimension is weighted by the inverse of the standard deviation of that component of the data vectors. e.g. c[12] would be weighted by (1/std_dev(c[12])). The standard deviations could be computed either over the entire data set (based on the global variance) or on a per-cluster basis (you use the standard deviation of each of the clusters you obtain during VQ to weight the distance from the mean of that cluster). Each of these two has a slightly different philisophy, and
could result in slightly different results.

A third thing you could do is to compute a Gaussian mixture with your data, and classify each data vector (or extended data vector, if you prefer to combine cepstra/dcep/ddcep into a single vector) as belonging to one of your gaussians. You then use the mean of that Gaussian as the codeword representing that vector. Dynamic ranges of data will not be an issue at all in this case.

Note: In the sphinx, for semi-continuous modeling, a separate codebook is made for each of the four feature streams: 12c,24d,3energy,12dd. Throughout the training, the four streams are handled independently of each other and so in the end we have four sets of mixture weights corresponding to each senone or hmm state. The sphinx does not do discrete modeling directly.

**Question:** For vector-quantization, should the control file entries correspond exactly to the transcript file entries?

**Answer** For the vq, the order of files in the ctl need not match the order of transcripts. However, for the rest of the training, the way our system binaries are configured, there has to be an exact match. The vq does not look at the transcript file. It just groups data vectors (which are considered without reference to the transcripts).

**Question:** What is the difference between the -stride flag in agg-seg and kmeans-init?

**Answer** -stride in agg-seg samples the feature vectors at stride intervals, the vectors are then used for VQ. In the kmeans-init program its function is the same, but this time it operates on the vectors already accumulated by agg-seg, so we usually set it to 1.

**Question:** Regarding the size of the VQ Codebook: is there something to say that the size 256 optimal? Would increasing the size affect the speed of decoding?

**Answer** For more diverse acoustic environments, having a larger codebook size would result in better models and better recognition. We have been using 256 codewords primarily for use with the SPHINX-II decoder, since for historical reasons it does not handle larger codebook sizes. The original sphinx-II used a single byte integer to index the codewords. The largest number possible was therefore 256. The format conversion code which converts models from SPHINX-III format to SPHINX-II format accordingly requires that your models be trained with a codebook size of 256.

The standard Sphinx-III decoder, however, can handle larger codebooks. Increasing the codebook size would slow down the speed of de-
coding since the number of mixture-weights would be higher for each HMM state.

**Question:** I am trying to do VQ. It just doesn’t go through. What could be wrong?

**Answer** It’s hard to say without looking at the log files. If a log file is not being generated, check for machine/path problems. If it is being generated, here are the common causes you can check for:

1. byte-swap of the feature files
2. negative file lengths
3. bad vectors in the feature files, such as those computed from headers
4. the presence of very short files (a vector or two long)
B.12 Updating existing models

**Question:** have 16 gaussian/state continuous models, which took a lot of time to train. Now I have some more data and would like to update the models. Should I train all over again starting with the tied mdef file (the trees)?

**Answer:** Training from the trees upto 16 or 32 gaussians per state takes a lot of time. If you have more data from the same domain or thereabouts, and just want to update your acoustic models, then you are probably better off starting with the current 16 or 32 gaussians/state models and running a few iterations of baum-welch from there on with *all* the data you have. While there would probably be some improvement if you started from the trees I dont think it would be very different from iterating from the current models. You *would* get better models if you actually built the trees all over again using all the data you have (since they would now consider more triphones), but that would take a long time.

**Question:** I have a set of models A, which have a few filler phones. I want to use additional data from another corpus to adapt the model set A to get a new adapted model set B. However, the corpus for B has many other filler phones which are not the same as the filler models in set A. What do I do to be able to adapt?

**Answer:** Edit the filler dictionary and insert the fillers you want to train. Map each filler in B to a filler phone (or a sequence of phones) in model set A. for example ++UM++ AX M ++CLICK++ +SMACK+ ++POP++ +SMACK+ ++HMM++ HH M ++BREATH++ +INHALE+ ++RUSTLE++ +INHALE+

On the LHS, list the fillers in B. On the RHS, put in the corresponding fillers (or phones) in A. In this case, it will be a many-to-one mapping from B to A.

To force-align, add the above filler transcriptions to the *main* dictionary used to force-align.
B.13 Utterance, word and phone segmentations

**Question:** How do I use the sphinx-3 decoder to get phone segmentations?

**Answer:** The decoder works at the sentence level and outputs word level segmentations. If your "words" are phones, you have a phone-decoder and you can use the -matchsegfn flag to write the phone segmentations into a file. If your words are not phones (and are proper words), then write out matchseg files (using the -matchsegfn option rather than the -matchfn option), pull out all the words from the output matchseg files *including all noises and silences* and then run a force-alignment on the corresponding pronunciation transcript to get the phone segmentation. You will have to remove the <s>, <sil> and /s/ markers before you force-align though, since the aligner introduces them perforce.

**Question:** How do I obtain word segmentations corresponding to my transcripts?

**Answer:** You can use the SPHINX decoder to obtain phone or word level segmentations. Replacing the flag -matchfn with -matchsegfn in your decode options will generate the hypotheses alongwith word segmentations in the matchfile. You can run a phone level decode in a similar way to obtain phone segmentations.

**Question:** The recordings in my training corpus are very long (about 30 minutes each or more). Is there an easy way to break them up into smaller utterances?

**Answer:** One easy way to segment is to build a language model from the transcripts of the utterances you are trying to segment, and decode over 50 sec. sliding windows to obtain the word boundaries. Following this, the utterances can be segmented (say) at approx. 30 sec. slots. Silence or breath markers are good breaking points.

There are other, better ways to segment, but they are meant to do a good job in situations where you do not have the transcripts for your recordings (eg. for speech that you are about to decode). They will certainly be applicable in situations where you do have transcripts, but aligning your transcripts to the segments would involve some extra work.
B.14 Force-alignment (Viterbi alignment)

**Question:** Will the forced-aligner care if I leave the (correct) alternate pronunciation markers in the transcript? Or do I need to remove them?

**Answer** The force-aligner strips off the alternate pronunciation markers and re-chooses the correct pronunciation from the dictionary.

**Question:** Some utterances in my corpus just don’t get force-aligned. The aligner dies on them and produces no output. what’s wrong?

**Answer** Firstly, let’s note that “force-alignment” is CMU-specific jargon. The force-aligner usually dies on some 1files. If the models are good, it dies in fewer cases. Force-alignment fails for various reasons - you may have spurious phones in your dictionary or may not have any dictionary entry for one or more words in the transcript, the models you are using may have been trained on acoustic conditions which do not match the conditions in the corpus you are trying to align, you may have trained initial models with transcripts which are not force-aligned (this is a standard practice) and for some reason one or more of the models may have zero parameter values, you may have bad transcriptions or may be giving the wrong transcript for your feature files, there may be too much noise in the current corpus, etc. The aligner does not check whether your list of feature files and the transcript file entries are in the same order. Make sure that you have them in order, where there is a one-to-one correspondence between the two files. If these files are not aligned, the aligner will not align most utterances. The ones that do get aligned will be out of sheer luck and the alignments will be wrong.

There may be another reason for alignment failure: if you are force-aligning using a phoneset which is a subset of the phones for which you have context-dependent models (such that the dictionary which was used to train your models has been mapped on to a dictionary with lesser phones), then for certain acoustic realizations of your phones, the context-dependent models may not be present. This causes the aligner to back up to context-independent (CI) models, giving poor likelihoods. When the likelihoods are too poor, the alignment fails. Here’s a possible complication: sometimes in this situation, the backoff to CI models does not work well (for various reasons which we will not discuss here). If you find that too many of your utterances are not getting force-aligned and suspect that this may be due to the fact that you are using a subset of the phone-set in the models used for alignment, then an easy solution is to temporarily restore the full phoneset in your dictionary for force-alignment, and once it is done, revert to the smaller set for training, without changing the order of the dictionary entries.
After Viterbi-alignment, if you are still left with enough transcripts to train, then it is a good idea to go ahead and train your new models. The new models can be used to redo the force-alignment, and this would result in many more utterances getting successfully aligned. You can, of course, iterate the process of training and force-alignment if getting most of the utterances to train is important to you. Note that force-alignment is not necessary if a recognizer uses phone-networks for training. However, having an explicit aligner has many uses and offers a lot of flexibility in many situations.

**Question:** I have a script for force-alignment with continuous models. I want to force-align with some semi-continuous models that I have. What needs to change in my script?

**Answer:** In the script for force-alignment, apart from the paths and model file names, the model type has to be changed from ".cont" to ".semi" and the feature type has to be changed to "s2_4x", if you have 4-stream semi-continuous.

**Question:** I’m using sphinx-2 force-aligner to do some aligning, it basically works but seems way too happy about inserting a SIL phone between words (when there clearly isn’t any silence). I’ve tried to compensate with this by playing with the -silpen but it didn’t help. why does the aligner insert so many spurious silences?

**Answer:** The problem may be due to many factors. Here’s a checklist that might help you track down the problem:

1. Is there an agc mismatch between your models and your force-aligner settings? If you have trained your models with agc "max" then you must not set agc to "none" during force-alignment (and vice-versa).

2. Listen to the words which are wrongly followed by the SIL phone after force-alignment. If such a word clearly does not have any silence following it in the utterance, then check the pronunciation of the word in your dictionary. If if the pronunciation is not really correct (for example if you have a flapped "R" in place of a retroflexed "R" or a "Z" in place of an "S" (quite likely to happen if the accent is non-native), the aligner is likely to make an error and insert a silence or noise word in the vicinity of that word.”

3. Are your features being computed exactly the same way as the features that were used to train the acoustic models that you are using to force-align? Your parametrization can go wrong even if you are using the *same* executable to compute features now as you used for training the models. If, for example, your training features were computed at the standard analysis rate of 100 frame/sec with 16khz,
16bit sampling, and if you are now assuming either an 8khz sampling rate or 8 bit data in your code, you’ll get twice as many frames as you should for any given utterance. With features computed at this rate, the force-aligner will just get silence-happy.

4. Are the acoustic conditions and *speech* bandwidth of the data you are force-aligning the same as those for which you have acoustic models? For example, if you are trying to force-align the data recorded directly off your TV with models built with telephone data, then even if your sampling rate is the same in both cases, the alignment will not be good.

5. Are your beams too narrow? Beams should typically be of the order of 1e-40 to 1e-80. You might mistakenly have them set at a much higher value (which means much *narrower* beams).
B.15 Baum-Welch iterations and associated likelihoods

**Question:** How many iterations of Baum-Welch should I run for CI/CD-untied/CD-tied training?

**Answer** 6-10 iterations are good enough for each. It is better to check the ratio of total likelihoods from the previous iteration to the current one to decide if a desired convergence ratio has been achieved. The scripts provided with the SPHINX package keep track of these ratios to automatically decide how many iterations to run, based on a “desired” convergence ratio that you must provide. If you run too many iterations, the models get overfitted to the training data. You must decide if you want this to happen or not.

**Question:** The training data likelihoods at the end of my current iteration of Baum-Welch training are identical to the likelihoods at the end of the previous iteration. What’s wrong and why are they not changing?

**Answer** The most likely reason is that for some reason the acoustic models did not get updated on your disk at the end of the previous iteration. When you begin with the same acoustic models again and again, the likelihoods end up being the same every time.

**Question:** The total likelihood at the end of my current Baum-Welch iteration is actually lower than the likelihood at the end of the previous iteration. Should this happen?

**Answer** Theoretically, the likelihoods must increase monotonically. However, this condition holds only when the training data size is constant. In every iteration (especially if your data comes from difficult acoustic conditions), the Baum-Welch algorithm may fail in the backward pass on some random subset of the utterances. Since the effective training data size is no longer constant, the likelihoods may actually decrease at the end of the current iteration, compared to the previous likelihoods. However, this should not happen very often. If it does, then you might have to check out your transcripts and if they are fine, you might have to change your training strategy in some appropriate manner.

**Question:** In my training, as the forward-backward (Baum-Welch) iterations progress, there are more and more error messages in the log file saying that the backward pass failed on the given utterance. This should not happen since the algorithm guarantees that the models get better with every iteration. What’s wrong?

**Answer** As the models get better, the “bad” utterances are better iden-
tified through their very low likelihoods, and the backward pass fails on them. The data may be bad due to many reasons, the most common one being noise. The solution is to train coarser models, or train fewer triphones by setting the "maxdesired" flag to a lower number (of triphones) when making the untied mdef file, which lists the triphones you want to train. If this is happening during CI training, check your transcripts to see if the within-utterance silences and non-speech sounds are transcribed in appropriate places, and if your transcriptions are correct. Also check if your data has difficult acoustic conditions, as in noisy recordings with non-stationary noise. If all is well and the data is very noisy and you can’t do anything about it, then reduce the number of states in your HMMs to 3 and train models with a noskip topology. If the utterances still die, you’ll just have to live with it. Note that as more and more utterances die, more and more states in your mdef file are "not seen" during training. The log files will therefore have more and more messages to this effect.

**Question:** My baum-welch training is really slow! Is there something I can do to speed it up, apart from getting a faster processor?

**Answer** In the first iteration, the models begin from flat distributions, and so the first iteration is usually very very slow. As the models get better in subsequent iterations, the training speeds up. There are other reasons why the iterations could be slow: the transcripts may not be force-aligned or the data may be noisy. For the same amount of training data, clean speech training gets done much faster than noisy speech training. The noisier the speech, the slower the training. If you have not force-aligned, the solution is to train CI models, force-align and retrain. If the data are noisy, try reducing the number of HMM states and/or not allowing skipped states in the HMM topology. Force-alignment also filters out bad transcripts and very noisy utterances.

**Question:** The first iteration of Baum-Welch through my data has an error:

```
INFO: ../main.c(757): Normalizing var

ERROR: "../gauden.c", line 1389: var (mgau=0, feat=2, density=176, component=1) < 0
```

Is this critical?

**Answer:** This happens because we use the following formula to estimate variances:

```
variance = avg(x2) - [avg(x)]2
```
There are a few weighting terms included (the baum-welch "gamma" weights), but they are immaterial to this discussion. The *correct* way to estimate variances is

\[
\text{variance} = \text{avg}[(x - \text{avg}(x))^2]
\]

The two formulae are equivalent, of course, but the first one is far more sensitive to arithmetic precision errors in the computer and can result in negative variances. The second formula is too expensive to compute (we need one pass through the data to compute \(\text{avg}(x)\), and another to compute the variance). So we use the first one in the sphinx and we therefore get the errors of the kind we see above, sometimes.

The error is not critical (things will continue to work), but may be indicative of other problems, such as bad initialization, or isolated clumps of data with almost identical values (i.e. bad data).

Another thing that usually points to bad initialization is that you may have mixture-weight counts that are exactly zero (in the case of semi-continuous models) or the gaussians may have zero means and variances (in the case of continuous models) after the first iteration.

If you are computing semi-continuous models, check to make sure the initial means and variances are OK. Also check to see if all the cepstra files are being read properly.
B.16 Dictionaries, pronunciations and phone-sets

Question: I’ve been using a script from someone that removes the stress markers in cmudict as well as removes the deleted stops. This script is removing the (2) or (3) markers that occur after multiple pronunciations of the same word. That is,

A EY
A AX

is produced instead of

A EY
A(2) AX

What is the consequence of removing this multiple pronunciation marker? Will things still work?

Answer: The (2), (3) etc. are important for the training. It is the only way the trainer knows which pronunciation of the word has been used in the utterance, and that is what the force-aligner decides for the rest of the training. So, once the force-alignment is done, the rest of the training has to go through with the same dictionary, and neither the pronunciations nor the pronunciation markers should change.

Independently of this, the script that you are using should be renumbering the dictionary pronunciations in the manner required by the trainer in order for you to use it for training and decoding. Pronunciation markers are required both during training and during decoding.

Question: I have trained a set of models, and one of the phones I have trained models for is “TS” (as in CATS = K AE TS). Now I want to remove the phone TS from the dictionary and do not want to retain its models. What are the issues involved?

Answer: You can change every instance of the phone “TS” in your decode dictionary to “T S”. In that case, you need not explicitly remove the models for TS from your model set. Those models will not be considered during decoding. However, if you just remove TS from the decode dictionary and use the models that you have, many of the new triphones in-
volving T and S would not have corresponding models (since they were not there during training). This will adversely affect recognition performance. You can compose models for these new triphones from the existing set of models by making a new tied-mdef file with the new decode dictionary that you want to use. This is still not as good as training explicitly for those triphones, but is better than not having the triphones at all. The ideal thing to do would be to train models without "TS" in the training dictionary as well, because replacing TS with T S will create new triphones. Data will get redistributed and this will affect the decision trees for all phones, especially T and S. When decision trees get affected, state tying gets affected, and so the models for all phones turn out to be slightly different.

**Question:** What is a filler dictionary? What is its format?

**Answer:** A filler dictionary is like any dictionary, with a word and its pronunciation listed on a line. The only difference is that the word is what *you* choose to call a non-speech event, and its pronunciation is given using whatever filler phones you have models for (or are building models for). So if you have models for the phone +BREATH+, then you can compose a filler dictionary to look like

```
++BREATHING++ +BREATH+
```

or

```
BREATH_SOUND +BREATH+
```

or...

The left hand entry can be anything (we usually just write the phone with two plus signs on either side - but that’s only a convention).

Here’s an example of what a typical filler dictionary looks like:

```
++BREATH++ +BREATH+
++CLICKS++ +CLICKS+
++COUGH++ +COUGH+
++LAUGH++ +LAUGH+
++SMACK++ +SMACK+
++UH++ +UH+
++UHUH++ +UHUH+
++UM++ +UM+
++FEED++ +FEED+
++NOISE++ +NOISE+
```

When using this with SPHINX-III, just make sure that there are no extra
spaces after the second column word, and no extra empty lines at the end of the dictionary.
B.17 Decision tree building and parameter sharing

**Question:** In HTK, after we do decision-tree-driven state-clustering, we run a "model compression" step, whereby any triphones which now (after clustering) point to the same sequence of states are mapped, so that they are effectively the same physical model. This would seem to have the benefit of reducing the recognition lattice size (although we’ve never verified that HVite actually does this.) Do you know if Sphinx 3.2 also has this feature?

**Answer** The sphinx does not need to do any compression because it does not physically duplicate any distributions. All state-tying is done through a mapping table (mdef file), which points each state to the appropriate distributions.

**Question:** The log file for bldtree gives the following error: INFO: ../main.c(261): 207 of 207 models have observation count greater than 0.000010 FATAL_ERROR: "../main.c", line 276: Fewer state weights than states

**Answer** The -stwt flag has fewer arguments that the number of HMM-states that you are modeling in the current training. The -stwt flag needs a string of numbers equal to the number of HMM-states, for example, if you were using 5-state HMMs, then the flag could be given as "-stwt 1.0 0.3 0.1 0.01 0.001". Each of these numbers specify the weights to be given to state distributions during tree building, beginning with the *current* state. The second number specifies the weight to be given to the states *immediately adjacent* to the current state (if there are any), the third number specifies the weight to be given to adjacent states *one removed* from the immediately adjacent one (if there are any), and so on.
B.18 Post-training disasters

Question: I’ve trained with clean speech. However, when I try to decode noisy speech with my models, the decoder just dies. shouldn’t it give at least some junk hypothesis?

answer Adding noise to the test data increases the mismatch between the models and test data. So if the models are not really well trained (and hence not very generalizable to slightly different data), the decoder dies. There are multiple reasons for this:

- The decoder cannot find any valid complete paths during decoding. All paths that lead to a valid termination may get pruned out.

- The likelihood of the data may be so poor that the decoder goes into underflow. This happens if even *only one* of your models is very badly trained. The likelihood of this one model becomes very small and the resulting low likelihood get inverted to a very large positive number because the decoder uses integer arithmetic, and results in segmentation errors, arithmetic errors, etc.

One way to solve this problem is just to retrain with noisy data.

Question: I’ve trained models but I am not able to decode. The decoder settings seem to be ok. It just dies when I try to decode.

answer If all flag setting are fine, then decoder is probably dying because the acoustic models are bad. This is because of multiple reasons a) All paths that lead to a valid termination may get pruned out b) The likelihood of the data may be so poor that the decoder goes into underflow. This happens if even *only one* of your models is very badly trained. The likelihood of this one model becomes very small and the resulting low likelihood get inverted to a very large positive number because the decoder uses integer arithmetic, and results in segmentation errors, arithmetic errors, etc.

You’ll probably have to retrain the models in a better way. Force-align properly, make sure that all phones and triphones that you do train are well represented in your training data, use more data for training if you can, check your dictionaries and use correct pronunciations, etc.

Question: I started from one set of models, and trained further using another bunch of data. This data looked more like my test data, and there was a fair amount of it. So my models should have improved. When I use these models for recognition, however, the performance of the system is awful. What went wrong?
answer The settings use to train your base models may have differed in one or more ways from the settings you used while training with the new data. The most dangerous setting mismatches is the agc (max/none). Check the other settings too, and finally make sure that during decoding you use the same agc (and other relevant settings like varnorm and cmn) during training.
Why is my recognition accuracy poor?

**Question:** I am using acoustic models that were provided with the SPHINX package on opensource. The models seem to be really bad. Why is my recognition accuracy so poor?

**answer** The reason why you are getting poor recognition with the current models is that they are not trained with data from your recording setup. While they have been trained with a large amount of data, the acoustic conditions specific to your recording setup may not have been encountered during training and so the models may not be generalizable to your recordings. More than noise, training under matched conditions makes a huge difference to the recognition performance. There may be other factors, such as feature set or agc mismatch. Check to see if you are indeed using all the models provided for decoding. For noisy data, it is important to enter all the relevant noise models (filler models) provided in the noise dictionary that is being used during decoding.

To improve the performance, the models must be adapted to the kind of data you are trying to recognize. If it is possible, collect about 30 minutes (or more if you can) of data from your setup, transcribe them carefully, and adapt the existing models using this data. This will definitely improve the recognition performance on your task.

It may also be that your task has a small, closed vocabulary. In that case having a large number of words in the decode dictionary and language model may actually cause acoustic confusions which are entirely avoidable. All you have to do in this situation is to retain *only* the words in your vocabulary in the decode dictionary. If you can build a language model with text that is exemplary of the kind of language you are likely to encounter in your task, it will boost up the performance hugely.

It may also be that you have accented speech for which correct pronunciations are not present in the decode dictionary. Check to see if that is the case, and if is, then it would help to revise the dictionary pronunciations, add newer variants to existing pronunciations etc. also check to see if you have all the words that you are trying to recognize in your recognition dictionary.

If you suspect that noise is a huge problem, then try using some noise compensation algorithm on your data prior to decoding. Spectral subtraction is a popular noise compensation method, but it does not always work.

All this, of course, assuming that the signals you are recording or trying to recognize are not distorted or clipped due to hardware problems in your setup. Check out especially the utterances which are really badly recognized by actually looking at a display of the speech signals. In fact, this is
the first thing that you must check.
B.20  Interpreting SPHINX-II file formats.

**Question:** (this question is reproduced as it was asked!) I was trying to read the SphinxII HMM files (Not in a very good format). I read your provided “SCHMM.format” file with your distribution. But, life is never that easy, the chMM_files format should have been very easy and straight forward..!!!

From your file ...

```
chmm FILES
```

There is one *.chmm file per ci phone. Each stores the transition matrix associated with that particular ci phone in following binary format. (Note all triphones associated with a ci phone share its transition matrix) (all numbers are 4 byte integers):

-10 (a header to indicate this is a tmat file)

256 (no of codewords)

5 (no of emitting states)

6 (total no. of states, including non-emitting state)

1 (no. of initial states. In fbs8 a state sequence can only begin with state[0]. So there is only 1 possible initial state)

0 (list of initial states. Here there is only one, namely state 0)

1 (no. of terminal states. There is only one non-emitting terminal state)

5 (id of terminal state. This is 5 for a 5 state HMM)

14 (total no. of non-zero transitions allowed by topology)

[0 0 (int)log(tmat[0][0]) 0] (source, dest, transition prob, source id)
There are thus 65 integers in all, and so each *.chmm file should be 65*4 = 260 bytes in size. ... that should have been easy enough, until I was surprised with the fact that the probabilities are all written in long (4 bytes) format although the float is also 4 bytes so no space reduction is achieved, Also they are stored LOG and not linear although overflow considerations (the reasons for taking the logs are during run time not in the files...)

All this would be normal and could be achieved .... but when I opened the example files I found very strange data that would not represent any linear or logarithmic or any format of probability values That is if we took the file "AA.chmm" we would find that the probabilities from state 0 to any other state are written in hex as follows:
As I recall that these probabilities should all summate to "1". Please, show me how this format would map to normal probabilities like 0.1, 0.6, 0.3 ...

**answer** First - we store integers for historic reasons. This is no longer the case in the Sphinx-3 system. The sphinx-2 is eventually going to be replaced by sphinx-3, so we are not modifying that system. One of the original reasons for storing everything in integer format was that integer arithmetic is faster than floating point arithmetic in most computers. However, this was not the only reason.

Second - we do not actually store *probabilities* in the chmm files. Instead we store *expected counts* (which have been returned by the baum-welch algorithm). These have to be normalized by summing and dividing by the sum. Finally - the numbers you have listed below translate to the following integers:

> (0,0) 00 01 89 C7 This number translates to 100807
> (0,1) 00 01 83 BF This number translates to 99263
> (0,2) 00 01 10 AA This number translates to 69802

These numbers are the logarithmic version of the floating point counts with one simple variation - the logbase is not "e" or 10; it is 1.0001. This small base was used for reasons of precision - larger bases would result in significant loss of precision when the logarithmized number was truncated to integer.

**Question:** The problem that I am facing is that I already have an HMM model trained and generated using the Entropic HTK and I want to try to use your decoder with this model. So I am trying to build a conversion tool to convert from the HTK format to your format. In HTK format, the transition matrix is all stored in probabilities!! So how do I convert these probabilities into your "expected counts".

**answer** You can take logbase 1.0001 of the HTK probs, truncate and store them.
B.21 Interpreting SPHINX-III file formats.

**Question:** what's up with the s3 mixw files? the values seem all over the place. to get mixw in terms of numbers that sum to one, do you have to sum up all mixw and divide by the total? any idea why it is done this way? is there a sphinx function to return normalized values? not that it's hard to write but no need reinventing the wheel... here's an example of 1gau mixw file, using printp to view contents:

```plaintext
——with -norm no
mixw 5159 1 1
mixw [0 0] 1.431252e+04 1.431e+04
mixw [1 0] 3.975112e+04 3.975e+04
mixw [2 0] 2.254014e+04 2.254e+04
mixw [3 0] 2.578259e+04 2.578e+04
mixw [4 0] 1.262872e+04 1.263e+04

-with -norm yes
mixw 5159 1 1
mixw [0 0] 1.431252e+04 1.000e+00
mixw [1 0] 3.975112e+04 1.000e+00
mixw [2 0] 2.254014e+04 1.000e+00
mixw [3 0] 2.578259e+04 1.000e+00
mixw [4 0] 1.262872e+04 1.000e+00
```

In s3, we have mixtures of gaussians for each state. Each gaussian
has a different mixture weight. When there is only 1 gaussian/state the mixture weight is 1. However, instead of writing the number 1 we write a number like "1.431252e+04" which is basically the no. of times the state occurred in the corpus. This number is useful in other places during training (interpolation, adaptation, tree building etc). The number following "mixw" like "mixw 5159" below merely tells you the total number of mixture wts. (equal to the total no. of tied states for 1 gau/st models). So

———with -norm no
mixw 5159 1 1
mixw [0 0] 1.431252e+04
1.431e+04
implies you haven’t summed all weights and divided by total and

———with -norm yes
mixw 5159 1 1
mixw [0 0] 1.431252e+04
1.0000..
implies you *have* summed and divided by total (here you have only one mixw to do it on per state), and so get a mixw of 1.
**B.22 Hypothesis combination**

**Question:** In the hypothesis-combination code, all the scaling factors of recombined hypotheses are 0. Why is this so?

**Answer:** Since the hypothesis combination code does not perform rescaling of scores during combination, there is no scaling involved. Hence the scaling factor comes out to be 0. This is usually what happens with any method that rescores lattices without actually recomputing acoustic scores. Second, the code uses *scaled* scores for recombination. This is because different features have different dynamic ranges, and therefore the likelihoods obtained with different features are different. In order to be able to compare different acoustic features, their likelihoods would have to be normalized somehow. Ideally, one would find some global normalization factor for each feature, and normalize the scores for that feature using this factor. However, since we do not have global normalization factors, we simply use the local normalization factor that the decoder has determined. This has the added advantage that we do not have to rescale any likelihoods. So the true probability of a word simply remains LM-score+acoustic score. The correct way of finding scaling factors (esp. in the case of combination at the lattice level, which is more complex than combination at the hypothesis level) is a problem that, if solved properly, will give us even greater improvements with combination.

**Question:** Given the scaling factor, the acoustic and LM likelihood of a word in the two hypothesis to be combined, how do we decide which one to be appear in the recombined hypothesis. For example, the word "SHOW" appears in both hypotheses but in different frames (one is in the 40th another is in 39th) - these words are merged - but how should we decide the beginning frame of word "SHOW" in the recombined hypothesis, and why does it become the 40th frame after recombination?

**Answer:** This is another problem with "merging" nodes as we do it. Every time we merge two nodes, and permit some difference in the boundaries of the words being merged, the boundaries of the merged node become unclear. The manner in which we have chosen the boundaries of the merged node is just one of many ways of doing it, none of which have any clear advantage over the other. It must be noted though that if we chose the larger of the two boundaries (e.g if we merge WORD(10,15) with WORD(9,14) to give us WORD(9,15)), the resultant merged node gets "wider". This can be a problem when we are merging many different hypotheses as some of the nodes can get terribly wide (when many of the hypotheses have a particular word, but with widely varying boundaries), resulting in loss of performance. This is an issue that must be cleared up for lattice combination.
**Question:** noticed that the acoustic likelihood of some emerging word doesn’t change from the original hypothesis to the recombined hypothesis. For example, the word “FOR” has acoustic likelihood to be -898344 in one hyp and -757404 in another hypothesis, they all appear in the 185th frame in both hyp. but in the recombined hyp, the word “FOR” appears at 185th frame with likelihood -757404, the same as in one of the hypotheses. These likelihoods should have been combined, but it appears that they haven’t been combined. Why not?

**Answer:** The scores you see are *log* scores. So, when we combine -757404 with -898344, we actually compute $\log(e^{-757404} + e^{-898344})$. But $e^{-757404} >> e^{-898344}$ as a result $e^{-757404} + e^{-898344} = e^{-757404}$ to within many decimal places. As a result the combined score is simply -757404.
B.23 Language model

**Question:** How can the backoff weights a language model be positive?

**Answer:** Here’s how we can explain positive numbers in place of backoff weights in the LM: The numbers you see in the ARPA format LM (used in CMU) are not probabilities. They are log base 10 numbers, so you have log10(probs) and log10(backoffweights). Backoff weights are NOT probabilities.

Consider a 4 word vocab

A B C D.

Let their unigram probabilities be

A 0.4
B 0.3
C 0.2
D 0.1

which sum to 1. (no [UNK] here).

Consider the context A. Suppose in the LM text we only observed the strings AA and AB. For accounting for the unseen strings AC and AD (in this case) we will perform some discounting (using whatever method we want to). So after discounting, let us say the probabilities of the seen strings are: P(A—A) = 0.2

\[ P(B|A) = 0.3 \]

So, since we’ve never see AC or AD, we approximate P(C—A) with

\[ P(C|A) = \text{bowt}(A) \times P(C) \]

and P(D|A) with

\[ P(D|A) = \text{bowt}(A) \times P(D) \]

So we should get

\[ P(A|A) + P(B|A) + P(C|A) + P(D|A) = \text{bowt}(A) \times (P(C) + P(D)) + P(A|A) + P(B|A) = \text{bowt}(A) \times (0.1 + 0.2) + 0.2 + 0.3 = 0.5 + 0.3 \times \text{bowt}(A) \]

But the sum P(A|A)…P(D|A) must be 1

So obviously \( \text{bowt}(A) > 1 \)

And \( \log(\text{bowt}(A)) \) will be positive.

bowts can thus in general be greater than or lesser than 1. In larger and fuller LM training data, where most n-grams are seen, it is mostly less than 1.
B.24  Training context-dependent models with untied states

**Question:** During the cd-untied phase, things are failing miserably. A very long list of “...senones that never occur in the input data” is being generated. The result of this large list is that the means file ends up with a large number of zeroed vectors. What could be the reason?

**Answer** The number of triphones listed in the untied model definition file could be far greater than the actual number of triphones present in your training corpus. This could happen if the model-definition file is being created off the dictionary, without any effective reference to the transcripts (e.g. minimum required occurrence in the transcripts = 0), and with a large value for the default number of triphones, OR if, by mistake, you are using a pre-existing model-definition file that was created off a much larger corpus.
B.25 Acoustic likelihoods and scores

**Question:** Acoustic likelihoods for words as written out by the decoder and (force-)aligner are both positive and negative, while they are exclusively negative in the lattices. How is this possible?

**Answer** The acoustic likelihoods for each word as seen in the decoder and aligner outputs are scaled at each frame by the maximum score for that frame. The final (total) scaling factor is written out in the decoder MATCHSEG output as the number following the letter "S". "T" is the total score without the scaling factor. The real score is the sum of S and T. The real score for each word is written out in the logfile only if you ask for the backtrace (otherwise that table is not printed). In the align output, only the real scores are written. The real scores of words are both positive and negative, and large numbers because they use a very small logbase (1.0001 is the default value for both the decoder and the aligner).

In the lattices, only the scaled scores are stored and total scaling factor is not written out. This would not affect any rescoring of a lattice, but might affect (positively or negatively) the combination of lattices because the scaling factors may be different for each lattice.

**Question:** In the following example

**Decoding:**

\[
\begin{align*}
\text{WORD} & \quad Sf \quad Ef \quad Ascore \quad Lmscore \\
\text{SHOW} & \quad 11 \quad 36 \quad 1981081 \quad -665983 \\
\text{LOCATIONS} & \quad 37 \quad 99 \quad -13693 \quad -594246 \\
\text{AND} & \quad 100 \quad 109 \quad -782779 \quad -214771 \\
\text{C-RATINGS} & \quad 110 \quad 172 \quad 1245973 \quad -608433 \\
\end{align*}
\]

**falign:** Sf Ef Ascore WORD

\[
\begin{align*}
11 \quad 36 \quad 2006038 \quad \text{SHOW} \\
37 \quad 99 \quad -37049 \quad \text{LOCATIONS} \\
100 \quad 109 \quad -786216 \quad \text{AND} \\
110 \quad 172 \quad 1249480 \quad \text{C-RATINGS}
\end{align*}
\]
We see that the score from decoding and falign are different even for words that begin and end at the same frames. Why is this so? I am confused about the difference in the ABSOLUTE score (the one without normalization by maximum in each frame) from decode and falign. In the above example, the absolute score for word "locations" (with lc "show", rc "and") beginning at frame no. 37 and ending at frame no. 99 is -13693 from the decode (I get the number from the decode log file, with backtrace on). while the score for exactly the same word, same time instants and same context is different in falign output (-37049). Can this be due to the DAG merge?

Answer There are several reasons why falign and decoder scores can be different. One, as you mention, is artificially introduced DAG edges. However, in the forward pass there is no DAG creation, so AM scores obtained from the FWDVIT part of the decode will not have DAG creation related artifacts. Other possible reasons for differences in falign and decode scores are differences in logbase, differences in beam sizes, differences in floor values for the HMM parameters etc. Even when all these parameters are identical the scores can be different because the decoder must consider many other hyotheses in its pruning strategy and may prune paths through the hypothesized transcript differently from the forced aligner. The two scores will only be identical if word, phone and state level segmentations are all identical in the two cases. Otherwise they can be different (although not in a big way). Unfortunately, the decoder does not output state segmentations, so you can't check on this. You could check phone segmentations to make sure they are identical in both cases. If they are not, that will explain it If they are, the possibility of different state segmentations still exists.

Question: The decoder outputs the following for utterance 440c0206:


Now let's say I have the same utts, and the same models, but a different processing (e.g. some compensation is now applied), and in this expt I get:

FWDXCT: 440c0206 S 36136210 T -25385610 A -21512567 L -394130
Let's say I want to see if this compensation scheme increased the likelihood of the utterance. Can I just compare the acoustic scores (after the "A") directly, or do I have to take the scaling ("S") into account somehow (e.g. add it back in (assuming it's applied in the log domain))? 

Answer You have to add the scaling factor to the acoustic likelihood against the A to get the total likelihood. You can then compare the scores across different runs with the same acoustic models.
B.26 Decoding problems

**Question:** I am trying to use the opensource SPHINX-II decoder with semi-continuous models. The decoder sometimes takes very long to decode an utterance, and occasionally just hangs. What could be the problem?

**Answer** Check your -agc*, -normmean and -bestpath flags. It is important to set the AGC/CMN flags to the same setting as was used to train the models. Otherwise, the decoder makes more mistakes. When this happens, when it tries to create a DAG for rescoring for the bestpath (which is enabled by setting "-bestpath TRUE") it gets trapped while creating the DAG and spends inordinate amounts of time on it (sometimes never succeeding at all). Even if the AGC/CMN flags are correct, this can happen on bad utterances. Set -bestpath FALSE and check if the problem persists for the correct AGC/CMN settings. If it does, there might be a problem with your acoustic models.
B.27 Why Sphinx III’s performance is poorer than recognizer X?

**Question:** Sphinx III’s default acoustic and language models appear to be not able to take care of tasks like dictation. Why?

**Answer** (By Arthur Chan at 20040910) Design of a speech recognizer is largely affected by the goal of the recognizer. In the case of CMU Sphinx, most of the effort were driven by DARPA research in 90s. The broadcast news models were trained in the so called eval97 task. Where transcription are required to be done for broadcast news. The above explains why the model don’t really work well for task like dictation. The data simply just for the use of dictation. Commercial speech application also requires a lot of specific tuning and application engineering. For example, most commercial dictation engine use more well-processed training material to train the acoustic model and language model. They also apply techniques such as speaker adaptation. CMU was very unfortunately don’t have enough resource to carry out these researches.
Bibliography


Index

sclite, 100
Alan Black, 23
Allen Newell, 14
Bayes Rule, 36
CALO, 21
CMU Communicator Toolkit, 29
CMU lex, 29
CMU Sphinx
   Definition, ix
CMUdict, 29
cmudict, 29
Data collection, 101
distance
   Euclidean, 46
   Mahalannobus, 47
distribution
   Gaussian, 47
Federick Jelinek, 14
Festival, 29
flat initialization, 137
Hephaestus, 28
Hierglyph
   Origin, ix
Hieroglyph
   writing process, x
Hieroglyphs, 25
James Baker, 14
John Pierce, 14
Kai-Fu Lee, 15
Kevin Lenzo, 23
License, 3
CMU-Cambridge LM Toolkit, 6
Sphinx II, 4
Sphinx III, 4
Sphinx IV, 5
SphinxTrain, 4
Off-line evaluation, 100
On-line evaluation, 99
OpenSALT, 30
OpenVXML, 30
probability, 32
   independence, 33
   conditional probability, 33
   joint probability, 33
   prior probability, 32
Raj Reddy, 14
Sentence Error Rate (SER), 100
definition, 100
Sphinx
   Sphinx I’s history, 15
   Sphinx II’s history, 16
   Sphinx III’s history, 18
   Sphinx IV’s history, 19
Sphinx Developers, ix
Sphinx2
   Sphinx 2.4, 8
   Sphinx 2.5, 7
Sphinx3
   Sphinx 3.4, 9
   Sphinx 3.5, 8
SphinxTrain, 22
   Decision Tree Manipulation
      Routines, 137
   Feature Extraction Routines, 137
Miscellaneous Routines, 137
Model Adaptation Routines, 137
Model Conversion Routines, 136
Model Estimation Routines, 137
Model Initialization Routines, 137
Model Manipulation Routines, 136
Software Architecture, 136

Transcription, 104

Word Error Rate (WER), 100
definition, 100